多家大型企业Oracle培训资深讲师倾心打造，深入浅出，化繁为简

- 视频讲解，实战教学
- 突出应用背景，内容贴近实际
- 全面围绕 Oracle SQL 常用技术展开
- ……
- 一本能快速上手的书
- 一本能解决实际问题的书
- 一本源自企业培训一线的教学资料

Oracle SQL

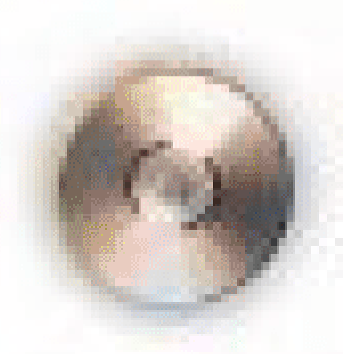
ORACLE DATABASE SQL

培训教程

——从实践中学习Oracle SQL及Web快速应用开发

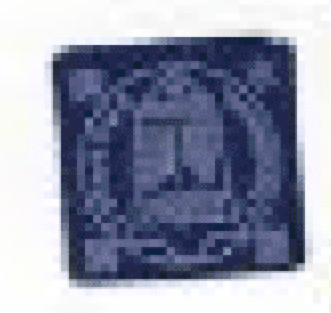
何明　何茜颖　等编著





DVD-ROM
30讲高清语音教学视频

技术支持：Sql_minghe@yahoo.com.cn
Liulm75@163.com



清华大学出版社

Oracle SQL 培训教程

——从实践中学习 Oracle SQL 及 Web 快速应用开发

何　明　何茜颖　等编著

清华大学出版社

北　京

内 容 简 介

本书是一本 Oracle SQL 的入门教材，它适合于初级到中级的读者。书中使用简单、生动的生活中的例子来解释复杂的计算机和数据库概念，而避免用计算机的例子。读者可以在没有任何计算机专业知识的情况下阅读此书。

本书又是一本 Oracle SQL 的实用教材，内容覆盖了 OCP（Oracle 认证专家）考试的几乎全部内容，但重点放在实际工作能力的训练。本书的每章中都有大量的例题，而且每道题都给出了答案。为了帮助读者理解，许多概念和例题都给出了商业应用背景，还有很多例题可以不加修改或略加修改即可应用于实际工作中。本书中的绝大多数例题都可以在 Oracle 8 及以上版本上运行。

本书所有图形操作和比较难的命令行操作都附有教学视频，读者可以在随书的光盘上找到，另外一些较难的命令已经被做成了正文或 SQL 脚本文件存在光盘上，读者可以通过复制和粘贴来运行它们。

图书在版编目（CIP）数据

Oracle SQL 培训教程——从实践中学习 Oracle SQL 及 Web 快速应用开发/何明，何茜颖等编著. —北京：清华大学出版社，2010.1

ISBN 978-7-302-21609-4

I. O…　II. ①何…　②何…　III. 关系数据库–数据库管理系统，Oracle–程序设计　IV. TP311.138

中国版本图书馆 CIP 数据核字（2009）第 219511 号

责任编辑：刘利民　朱　俊
版式设计：魏　远　王世月
责任校对：王　云
责任印制：
出版发行：清华大学出版社　　**地　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：**100084
社 总 机：010-62770175　　**邮　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：185×260　**印　张：**38　**字　数：**872 千字
（附 DVD 光盘 1 张）
版　次：2010 年 1 月第 1 版　**印　次：**2010 年 1 月第 1 次印刷
印　数：1～5000
定　价：69.80 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：035520-01

鸣　　谢

首先，我要感谢新西兰 GZ Comtech（NZ）LTD（现为 New Zealand Institute of Science and Technology Ltd，新西兰科学技术学院）和 Unitec Institute of Technology（新西兰奥克兰技术学院）的同事和学生们，他们对产生本书初稿的讲义和例题提出了许多宝贵的意见。

其次，我要感谢中国计算机软件与技术服务总公司培训中心、昆仑瑞通高级技术培训中心和中国 UNIX 协会的同事和学生们，特别是那些阅读了本书部分初稿并给出了反馈意见的学生们。我也要感谢一切支持和关心过本书写作的朋友、同事和学生们。

在这里我要特别感谢的是清华大学出版社的刘利民编辑，这本书是在他的鼓励和支持下完成的。在本书的策划和写作期间一直得到他的支持，他为本书的写作提出了许多宝贵的意见。

最后，我必须感谢我的家人，感谢她们的支持和理解，特别是我的两个女儿，我用去了许多应该属于她们的时间来完成这本书的写作。

序

这本书将帮您推开 Oracle 那扇紧闭的大门，引领您登上数据库应用系统开发或管理的方舟。

由于信息量的不断增加，目前应用系统一般都要使用一个强大的后台数据库来存储和管理系统所需的大量数据，而数据的展示往往使用互联网网页的方式。本书由两部分组成，第一部分是关于 Oracle 的 SQL 语言，第二部分是关于 Oracle 新推出的快速 Web 开发工具。由于 Oracle 的 SQL 本身是 SQL 国际标准（ISO）的超集，同时，在本书中介绍的 Oracle 开发工具又可以直接访问其他数据库（如 MySQL、SQL Server 和 Access）的数据，所以即使读者从事其他数据库应用系统的开发或管理也将从中受益。

本书是在 2004 年清华大学出版社出版的《从实践中学习 Qracle SQL》一书的基础上修订而来。

自从《从实践中学习 Oracle SQL》一书出版以来，收到了许多读者（包括教学和培训机构）的反馈，有些读者还提出了一些宝贵的建议，在此表示衷心的感谢。

《从实践中学习 Oracle SQL》一书系统而详细地介绍了 Oracle 的 SQL 和 SQL*Plus，使用书中所介绍的 SQL 和 SQL*Plus 命令就可以进行数据库的开发、管理及维护工作。但是接下来可能出现另一个问题，那就是如何使那些没有任何数据库或计算机知识和背景的用户在未经培训的情况下就能方便地使用这些数据，显然使用 SQL 和 SQL*Plus 命令对这样的人群是极为困难的。可能有的读者已经想到了网页，让用户使用在互联网上冲浪的方法访问和使用这些数据应该是最佳的方案，也是眼下最时尚的。

在 Oracle 10g 之前将 Oracle 数据库中的数据做成网页放在互联网上并不轻松，因为使用 SQL 不能进行直接的网页编程，而必须借助于其他的程序设计语言或工具，而且在使用这些语言或工具之前经常不得不完成一些繁琐的系统配置。Oracle 11g 的 Oracle 快速 Web 应用开发工具的诞生将使这些令人不愉快的“噩梦”成为历史。使用这一图形化的工具只需一些简单的鼠标点击、拖拉或极少的输入就可以将 Oracle 数据库中的数据以优美的网页轻松地展示给用户。

Oracle 快速 Web 应用开发工具的前身是 Oracle 10g 的 HTML DB。当我在几年前第一次接触到 Oracle 10g 的 HTML DB 时曾为之一震，因为使用这一图形工具可以在没有或只有很少 SQL 知识的情况下进行互联网应用系统的开发和部署，它可以直接将 Oracle 的表或视图（甚至查询结果）的数据以网页形式展示给用户，而这些数据与数据库表中的数据是关联的，即当表中的数据发生变化时，相关网页中的数据也随之自动刷新。

我当时就想将 Oracle 10g 的 HTML DB 的内容加到《从实践中学习 Oracle SQL》书中，以方便读者迅速地将自己使用 SQL 语言在 Oracle 数据库中存储的数据以优美的网页形式展示给用户，但是很快就放弃了这个念头，因为 Oracle 10g 的 HTML DB 的安装相当复杂，另外，当时所需的内存等系统资源也太多。

Oracle 公司在推出 Oracle 11g 时，将 HTML DB 改名为 Oracle 快速 Web 应用开发（Oracle Application Express）。当我在 2008 年下半年拿到 Oracle 快速 Web 应用开发 3.12 时，发现它已经比之前的版本有了相当大的改进，其中最主要的是它已经嵌入在 Oracle 11g 数据库中，其安装和配置也简化了许多，另外，由于计算机硬件价格的持续下降和性能的不断提高使得它的硬件资源消耗问题也变得无关紧要了。

Oracle 快速 Web 应用开发工具是一种声明类型的开发工具，它适合于以数据库为中心的互联网应用程序的开发与部署。只需要使用网络浏览器和很少的（甚至没有）编程经验，用户就可以利用这一工具开发出快捷、安全，同时具有专业水平的应用程序。

实际上，Oracle 公司已经将它以前的开发工具的表单（Form）、报表（Report）和图表（Graph）等功能集成到了 Oracle 快速 Web 应用开发工具中。使用这一工具，一个开发人员在一周甚至几天内就可以完成许多程序员数月才能完成的应用系统开发（使用传统程序设计语言），因此，这一工具在竞标应用系统开发项目时非常有用，因为读者可以很快地将系统未来的界面和操作方式演示给用户，无疑增加了中标的可能性。

本书在原书《从实践中学习 Oracle SQL》的基础上进行了扩展，其增加的主要内容是第 18~23 章，主要讲解利用 Oracle 快速 Web 应用开发工具开发和部署基于 Oracle 数据库的互联网应用程序。同时也对 Oracle 10g 和 Oracle 11g 变化的内容做了相应的修改。为了增加趣味性和帮助读者更好地理解商业背景及实际应用，在这部分使用了一个叫“武大郎烧饼总公司”的虚拟公司，本书将详细地介绍如何使用 Oracle 快速 Web 应用开发工具为“武大郎烧饼总公司”开发和部署一个基于 Oracle 数据库的互联网应用系统。

本书使用的是 2009 年 2 月才推出的 Oracle 快速 Web 应用开发 3.2 版。为了简化这一工具的安装和配置，本书将安装和配置所需的命令及参数放入了一个正文文件，读者可以直接使用复制和粘贴来快速方便地完成 Oracle Application Express 的安装与配置。另外，为了减轻读者学习的难度，本书还对一些章节配备了视频教程（在随书的 DVD 光盘中）。在这个光盘中还包括了一些比较复杂例题的脚本文件，如果读者不想输入复杂和冗长的命令，就可以直接运行这些脚本文件或使用复制和粘贴的方法来执行这些命令。

本书除了介绍 Oracle 快速 Web 应用开发的内容之外，还增加了如下内容：

- 数据的集合运算。
- Oracle 新推出的免费图形工具 SQL Developer 和第三方的图形工具。
- Oracle 与其他软件（如 BI）的数据交换。
- Rownum 伪列的特殊用途。
- 永久配置 SQL*Plus 参数和替代变量等。

由于本书进行了精心的设计，读者完全可以根据需要先学习第二部分。如果作为教材使用，老师完全可以根据课程安排只讲授其中的一部分，而将另一部分作为参考内容使用。

除了有志成为 Oracle 专业人士的读者外，本书也适合那些没有 IT 背景的读者在比较短的时间内掌握基于数据库的互联网应用程序的开发，这样，公司的一些不太重要或使用时间不长的应用程序就不用雇用软件公司来开发了，读者学完本书后完全可以胜任这些开发工作。当然，如果读者已经掌握了 Oracle 的 PL/SQL 程序设计语言或 Java，就完全可以开发出复杂而高效的应用系统，因为 Oracle Application Express 可以直接使用 PL/SQL 和

Java 的程序。

此外，为了让读者更快、更容易地学习和获取他人的知识和成果，本书还新增加了逆向工程的内容（尽管该部分不属于 Oracle 标准培训的内容），即如何使用图形工具导出他人的设计（E-R Model）以及如何导出他人的源程序等，即教读者怎样做“贼”和介绍“贼”常使用的图形工具。

我们的祖先之所以能从灵长类中脱颖而出进化成万物之灵的人类，就是因为学会了使用和发明工具。借助于 Oracle 的 SQL 和 Application Express 这些强大的工具，相信那些只有很少，甚至没有 IT 背景的读者也会轻松而迅速地从“菜鸟”进化成“大虾”，再进化成“专家”和“大师”，最后在退休时进化成“一代宗师”。

何　明

前　　言

SQL（Structured Query Language）是标准的关系数据库（Relational Database Management Systems）操作语言。它是一种非过程化的第 4 代高级语言，其语法与英语非常相似，因此，也是一种比较容易学习的计算机语言。SQL 操作语言对初学者几乎没有任何要求，换句话说，初学者可以不具备任何计算机经验。

Oracle 是一个适合于大中型企业的数据库管理系统，其市场占有率是所有的数据库管理系统中最高的，它主要的用户为银行、电信、移动通信、航空、保险、金融、跨国公司和电子商务等。根据 WTO 的有关协议，我国在以上领域要逐年开放市场，因此，随着这些领域外资的大量涌入，在不远的将来对 Oracle 数据库管理员和开发人员的需求将急剧增加。

Oracle SQL 是 Oracle 数据库管理系统上的 SQL 语言，无论是 Oracle 数据库管理员（DBA）、开发人员（Developer）还是一般用户，熟练地使用 Oracle SQL 都是最基本的要求。OCP（Oracle Certified Professional）认证考试也将 Oracle SQL 作为 Oracle 数据库管理员（DBA）和开发人员（Developer）的第一门必考课程。

Oracle SQL 是美国国家标准化委员会（American National Standards Institute，ANSI）和国际标准化组织（International Standards Organization，ISO）颁布的 SQL 标准的超集。也就是说，如果读者学会了 Oracle SQL，就能很容易地掌握其他数据库管理系统上的 SQL 语言。

目前国内已出版的 Oracle 教材以翻译书为主，许多是针对 OCP 考试的，其内容的编排多是以总复习的形式出现的。这些教材应付 OCP 考试有用，但不完全适合作为培训教材，更不适于自学。Oracle Student Guide （英文）虽然是一套很好的培训教材，但这套书是不外卖的，只有参加 Oracle 公司的培训才能得到，而它的培训费之高一般人很难负担得起。

Oracle Student Guide 的另一问题是：它的练习题对系统资源要求很高，例如，它要求学生创建的表或表空间常常以百兆字节为单位，另外，它的许多练习题必须使用 Oracle 提供的脚本文件才能运行。也就是说，如果不参加 Oracle 公司的培训，学生是很难实现许多练习题的。

这本书是源于我在新西兰 GZ Comtech（NZ）LTD（现为：New Zealand Institute of Science and Technology Ltd——新西兰科学技术学院）从事 Oracle 数据库管理员（DBA）培训课程时的讲稿和为学生编写的上机题。当时为了使学生能有足够的上机练习机会，我几乎为每一讲都设计了大量的上机练习题，为了使学生能在家里练习这些上机题，每一道题都进行了精心的设计，使其对系统的资源消耗都限制在一般 PC 允许的范围（当时 PC 的一般硬件配置为：CPU 300 MHz 左右，内存 64MB，硬盘 8GB 左右），而且这些上机题完全是自封闭的，即学生不需要任何 CD，不需要运行任何脚本文件，除 Oracle 系统外也不需要安装任何其他的软件。

之后，这些讲稿和上机练习题曾在多个培训机构的多种 Oracle 培训课程上使用。其中包括中国计算机软件与技术服务总公司培训中心（北京）、昆仑瑞通高级技术培训中心（北

京）、中国 UNIX 协会和 Oracle 大学（Etake Technology Inc）等。部分讲稿也在新西兰的 Unitec Institute of Technology - 新西兰奥克兰技术学院（公立）为大学本科生讲授数据库概论时使用过多次。同时也从学生和同事们那里得到了大量有益的反馈。许多学生本身就是工作在电信、移动通信和航空等大型企业的数据库管理员或数据库开发人员，他们提出了很多在工作中遇到的实际问题，这些实际问题及其解决方案后来也都加到了讲稿或上机题中。一些我做信息系统管理员、分析员和 IT 顾问时所遇到的实际问题和解决方案也逐步地加到了该讲稿或上机题中。

考虑到参加培训的许多学生正在工作，而且一些学生基本上没有计算机背景，他们或者是没时间或者是没能力来理解难懂而乏味的学术术语，本书使用生动而简单的生活中的例子来解释复杂的计算机和数据库概念，而避免用计算机的例子。所以本书对学生的计算机专业知识几乎是没有任何要求。对以前培训的学生的跟踪表明这样的设计是合理的。

书中许多概念和例题都给出了商业应用背景，许多例题是用场景或故事的形式出现的。不少例题和它们的解决方案是企业中的数据库管理员或数据库开发人员在实际工作中经常遇到或可能遇到的，因此很多例题可以不加修改或略加修改后应用于实际工作中。

本书的内容和例题设计由浅入深，为了消除初学者对计算机教材常有的畏惧感，本书把那些难懂而且又不常用的内容尽量放在书的后面章节或附录中，并去掉了个别非常难懂而且一般的 Oracle 工作人员都很少听到的内容。根据我多年的 IT 工作和教学经验，一个人在某个系统中所使用的功能是很少的，相信还不到一半。因为绝大多数难懂的操作可以通过其他操作的组合来实现，因此，没有必要为了解释清楚 5%非常难懂的内容而吓跑了 95%的读者。

本书是一本 Oracle SQL 的实用教材。虽然它覆盖了 OCP 这部分考试的几乎全部内容，但重点放在实际工作能力的训练。该书全面而详细地介绍了关系数据库（Relational Database Management Systems）的标准操作语言和 SQL（Structured Query Language）语言，它包括数据查询语言、DML（数据操作语言）、DDL（数据定义语言）、DCL（数据控制语言）和事务处理。该书也详细地介绍了常用的 SQL 函数及 Oracle 8i 和 Oracle 9i 所提供的一些高级功能。

在刚开始写这本书时，我曾经想为读者构造一个小型的订单系统，并以光盘的形式随书赠送给读者。但是经过仔细权衡利弊之后终于打消了这个念头。因为如果这样做的话，就要求读者在开始做练习之前先学会安装光盘的内容，这样势必会增加读者开始学习的难度。考虑到不少读者可能是初学者这一事实，最后还是决定在开始时使用 Oracle 系统自带的几个表，随着学习的深入，再教读者自己构造一些所需要的表和其他的对象。

学习这门课的许多人可能是第一次接触 Oracle，对如何设置 Oracle 环境一无所知，所以在计算机上配置 Oracle 的运行环境的操作越简单越好，这正是本书为什么选用在 Windows 环境下运行 Oracle 数据库管理系统的主要原因。

选用 Windows 作为学习环境的另一个原因是学习的成本。因为 Windows 是一个相对便宜的操作系统，而且它很容易得到。绝大多数人对这一操作系统都有所了解，因此在学习 Oracle 之前不需要再学习操作系统。

Oracle 数据库管理系统是独立于任何 IT 平台的，所以当在一个操作系统上学会了 Oracle 的 SQL 语言的使用之后，就可以在任何操作系统上使用它了。也可以把 SQL 语句或脚本文件几乎不做任何修改地从一个操作系统移植到另一个操作系统上。另外，尽管许

多大型的 Oracle 数据库系统的服务器是安装在 UNIX 或其他操作系统上，但是用户的前台终端还是使用 Windows 操作系统。

本书的每一章中都配有大量的例题。从我的工作和教学实践中得来的经验表明：自己上机做练习是一种很好的学习方法，这样做往往比只看书效果好。数据库是一门实践性很强的课程，只有通过大量的上机操作实践（练习）才能悟出 SQL 语言的真谛，水平才能上到一个新的层次。正像毛主席说的“要在游泳中学会游泳”。如果不跳到水中就永远学不会游泳，如果不坐在计算机前真正地操作 Oracle 数据库系统，是很难真的学会使用 Oracle 数据库系统的，因此建议读者如果有条件最好把本书中的例题在自己的计算机上重做一遍或多遍。

本书中几乎每一道例题都给出了显示结果，其目的有两个，第一，当读者重做例题后，这个显示结果可以帮助读者检查所做的是否正确；第二，如果读者根本就没有能力买一台计算机，则这些显示结果可以帮助读者更好地理解书中的内容。

当阅读本书时，会发现书中没有指定的练习题，这是因为每一章有很多例题的缘故。读者只要把这些例题重做 1~2 遍也就达到了练习的目的。另外，本书在每章的结尾处并未给出思考题，而使用了“您应该掌握的内容”这样的方式，之所以没有使用思考题这个词是为了避免束缚读者的想象力。使用“您应该掌握的内容”这样比较宽松的句子的好处是：当思考所列出的内容时，只要已经理解了它们就可以了，至于如何解释和回答它们已经变得不重要了。

在我的教学实践中经常遇到这样的情况：有的学生看到书中的讨论或解释时就觉得很吃力，有时甚至想睡觉，但是上机做练习时他们马上就精神起来，而且做得还挺好。如果这种情况有时也适合您的话，请不用担心，只要能理解书中所介绍的内容就达到了目的，至于是通过上机做练习还是通过阅读书中的解释学会的并不重要。还是那句话“不管白猫黑猫，抓住老鼠就是好猫”。科学已经证明，文字作为一种交流的工具，它的承载能力要比声音和图像小，这可能也是为什么提倡多媒体教学的原因之一。所以当您看书时，有些内容看一遍看不懂是很正常的。

如果真的有一段内容看了几遍都不能完全理解，也用不着害怕，可以先把这段内容跳过，继续下面的学习。因为本书的编排不是严格的一环扣一环的。等看完了几章之后，回过头来再看这段内容可能就比较容易理解了。

书作为一种古老的单向交流工具，它的承载能力是很有限的，因此产生二义性几乎是不可避免的。为了减少二义性的产生，我曾把本书中许多章的初稿分别发给了多个培训机构的学生们，并根据他们阅读后反馈回来的意见对相关的章节做了相应的修改，其中，有些章节几乎是全部重写。尽管做了这些努力，但也很难保证该书像武侠小说或爱情小说那样容易理解，因为它毕竟不是一本消遣的书。

记得我在做硕士论文时，我的导师一再嘱咐在论文中要避免使用第一人称（我）和第二人称（您），要尽量避免口语化，最好使用被动语态。我在写论文时确实是那样做的。不是我喜欢，因为不那样做可能就毕不了业。

但是这本书却使用了不少口语，因为这本书的读者不少是初学者而不是专家。使用口语的目的主要是为了减少初学者学习的难度。本书在解释 SQL 语句或概念时没有追求学术上的完美，而只是给出了实用的解释。在本书中甚至没有使用语法这个词。

在本书中有不少虚构的故事，在这些故事中使用了不少夸张性的语言，其目的只是增加读者的兴趣。因为我深知看有用的书时多数人都很容易产生睡意，使用这些夸张性的语言可能会使读者在阅读时不至于睡着。

本书中使用的“您”并不是指读者。这样的写法是想让读者在阅读此书时尽量地投入进去。读者可以把书中所讲的故事看成一出戏，而您正是这出戏中的一个主角。这样或许对您理解书中所介绍的内容会有所帮助。书中常提到的“您的老板”、“您的经理”和“您的上司”等都不是现实中的人。相信现实中的他们可能都是大好人、大善人、灰衣天使、黑衣天使……

许多人认为学习 IT，特别是学习 Oracle 数据库管理系统是既枯燥又令人生畏的。一些人下了决心去学 Oracle，他们可能用几个月的时间拼命地学完了 Oracle 公司所要求的课程并通过了考试，在这一段时间里没有什么娱乐。等拿到证书后就把所有的书都丢到一边，痛痛快快地放松一下一直绷紧的神经。

希望这本书的写法能在枯燥的 Oracle 学习与娱乐之间达到某种程度的平衡，从而不至于使读者在整个学习过程中神经一直绷得很紧。

参加应试培训可能出现的另一个问题是：有些学生拿到了证书之后还不能进行熟练的上机操作。有的培训中心的课程基本上只进行应试培训，只教学生如何做考题，这样学生的动手能力就比较差，在应聘或面对实际工作时就将处在一种不利的境地，因为没有哪个公司愿意请一个只会说而不会干的人。最后检验学生能力的还是市场而不是考场。

这本书对以上的这些人会有很大的帮助，因为这本书的重点是放在了能力的培养，即教您怎样干活。一个既会说又会干的人一定是市场上最需要的人才。

本书既可作为企业或培训机构的 Oracle SQL 课程的培训教材，也可作为自学教材。

编写这本书的目的有如下 3 个：

- 把那些没有计算机或 Oracle 背景但想加入 IT 产业的人带入 Oracle 这个就业市场中来。
- 为那些有计算机或 Oracle 经验但没受过 Oracle 正规培训的人提供一套系统而完整的 Oracle 培训教材。
- 为那些非计算机人员，如管理或行政人员，了解和使用 Oracle 提供一套完整易学的培训教材。

本书中的绝大多数例题都分别在 Oracle 的 8.0.4、8.0.5、8.1.5 和 8.1.7 等版本上测试过，所有的例题都在 Oracle 的 9.0.1 版本上测试过，所以对您所使用的 Oracle 版本几乎没什么要求。

参与本书编写和资料整理的有王莹、万妍、王逸舟、牛晨、王威、程玉萍、万群柱、王静、范萍英、范秀英、汪超英、汪洁英、汪莉、黄力克、万洪英、万节柱、万如更、李菊、万民柱、万晓轩、赵京、张民生和杜蘅等，在此对他们辛勤和出色的工作表示衷心的感谢。

如果读者对本书有任何意见或要求，欢迎来信提出。我的电子邮箱为 sql_minghe@yahoo.com.cn。最后，预祝读者的 Oracle SQL 学习之旅轻松而愉快！

编　者

导　　读

不少读者在自学过程中常有把每一章中的练习题都做完了再学习下一章的习惯，如果有一两道题不理解，他们可能花费大量的时间去试着找出合理的解释。这样做的结果可能会因为少量的几道题不理解而影响了整本书的学习，而这几道题也许在以后的工作中根本就用不到。

为了避免这种情况的发生，本书没有给出习题，取而代之的是给出了大量的例题。因为每一个例题都有具体的操作步骤和结果，绝大多数还有解释。如果读者没有看懂某一个例题，按照给出的操作步骤在计算机上做一遍一定会对读者的理解有所帮助，也就是所谓的“照葫芦画瓢”。

对每章最后的“您应该掌握的内容”（相当于思考题）只要理解就可以了，至于如何解释和回答并不重要。如果觉得某个思考题或例题特别难理解，可以先跨过去，继续后面的学习，千万不要在一道题或一段内容上“打转”，浪费过多的时间。

本书的前 9 章都不长，其中，第 1 章、第 2 章、第 3 章、第 6 章和第 8 章的长度还不足 20 页。之所以这样安排，是因为如果读者能在较短的时间里学会一章的内容，多数都会有一种成功感，这样可以激励他们继续学下去。这些章节中的不少内容在后面的章节中都有重复，等读者掌握了这些基本内容后，就会有一定的经验，自信心也会有所增强，这时再阅读较长的章节，就不会有初学者阅读 IT 书籍时常有的恐惧感了。

如果读者对命令行界面和操作理解有困难，只需对第 3 章的内容有一个大概了解就可以了，不用花太多的时间，因为在 Windows 上运行的 SQL*Plus 也提供了不少图形操作的功能。

第 4 章是介绍单行函数的。读者在阅读这一章时，也只要对该章的内容有一个大概了解就可以了，不用在每一个函数上花过多的时间，等将来用到哪个函数时再具体查阅。如果读者在理解日期的 RR 格式或 YY 格式时有困难，也不要在上面花太多的时间，只要记住尽量使用 4 位数表示年就可以了。

第 1 章和第 2 章的内容要仔细地查阅，其中的例题最好能在计算机上做 1~2 遍，因为这两章的内容是数据查询（SELECT）语言最基本的内容。如果读者不能彻底理解这些内容，后面的学习将会非常困难。

学习第 5 章时要多花一些时间，应把该章中的例题都在计算机上做 1~2 遍。空值（NULL）看上去很容易理解，但使用时极容易出错，而且这种错误不属于语法错误，系统在编译 SQL 语句时检查不出任何错误。尽管许多同类书籍中对空值（NULL）的讨论并不多，但不少含有 SQL 的应用程序的错误都是由于空值（NULL）处理不当造成的。

第 6 章也是一个难点，希望读者能把这一章的内容理解透彻，并把该章上的例题都在计算机上做 1~2 遍。虽然分组函数可以为管理者或决策者提供丰富的信息，但如果使用不当，它们很容易产生错误，而且它们对系统效率的冲击也是不容忽视的。

如果读者没有数据库或计算机方面的学习经历，在阅读第 7 章时，可以把主要的精力放在 Oracle 提供的相等连接（Equijoin）、自连接（Self join）、不等连接（Non-equijoin）和外连接（Outer join）这 4 种类型的连接上。读者即使没有掌握 SQL:1999 语法连接语句，也不会影响在实际工作中使用 SQL，而且 SQL:1999 语法连接语句并没有提供效率方面的好处。

对于初学者来说，第 8 章中有些内容可能并不容易理解。读者可以在第一遍阅读时采取读懂多少算多少的策略，之后继续后面各章的学习，因为后面的内容与这一章几乎没什么直接的关系。等读者有一定的 SQL 的实践经验时再重新阅读这部分的内容就很容易理解了。

阅读第 9 章时不用花过多的工夫，只要对该章的内容有一个大概了解就可以了。读者可以把这一章看成手册，等将来用到时知道到哪查阅。因为在市场上可以买到许多图形开发工具，如 Oracle 的 FORM 和 REPORT 等，使用这些图形开发工具可以更容易地产生优美的和友好的提示或输出。关于 Oracle 的数据字典，只有理解了 Oracle 体系结构之后，才能真正地理解它们。

第 11 章所介绍的替代变量对开发应用程序很有用。读者在阅读这一章时要在 ACCEPT 命令的用法上多下些工夫，因为使用该命令可以比较容易地开发出用户友好的应用程序。但是如果您只是一个普通用户而且也不在数据库上开发应用程序，就不用花过多的工夫，只对该章的内容有一个大概的了解就可以了。

如果读者只是一个普通用户，而且也不在数据库上开发应用程序，只需要熟练掌握第 1 章、第 2 章、第 5 章、第 6 章和第 7 章中 Oracle 提供的 4 种类型的连接就可以工作了。

如果读者是一个普通用户但是要在数据库上开发应用程序，除了需要熟练掌握以上的内容之外，还需要熟练掌握第 3 章中有关脚本文件的内容、第 4 章中一些常用的函数、第 7 章中有关数据库规范化的内容、第 8 章中的单行子查询、第 9 章和第 11 章中有关控制 SQL*Plus 的环境的内容。

第 10 章开始介绍 DDL 语言。如果您是一个开发人员（程序员）或需要在数据库中创建表，这一章的内容要仔细地阅读，因为这一部分是 DDL 语言最基本的内容。对于其他的数据库人员来讲，只要能基本掌握本章的内容，甚至只是大概地了解就可以了。

第 12 章开始介绍数据操作语言（DML）和事务控制（Transaction Control）。如果您是一个开发人员（程序员）或需要修改数据库中的数据，这一章的内容要仔细地阅读，因为只有 DML 操作才能改变数据库中的数据。对于其他的数据库人员来讲，只要能基本上掌握本章的内容，甚至只要大概地了解就可以了。

第 13 章的内容对数据库管理员和开发人员（程序员）来说很有用。数据库管理员在对数据库系统性能进行优化时可能用到这一部分的内容。这一部分的内容对开发人员（程序员）尤为重要，如果没有彻底理解这一部分的内容，是很难开发出“像样”的应用程序（基于数据库系统的）的，即使勉强开发出来了，也很难维护。

如果您想成为数据库管理员，就要认真学习第 14 章的内容。因为数据库管理员在对数据库系统性能进行优化时也可能用到这一部分的内容。如果您想成为开发人员（程序员），也应该基本上掌握本章的内容，这样就不会开发出一些莫名其妙的或运行效率很低的应用程序（基于数据库系统的）。

如果您想成为开发人员（程序员），就应该熟练掌握第 15 章的内容。这一部分的内容对开发实际的商业数据库系统可能很有帮助。对于其他的数据库人员，只要能基本上掌握本章的内容，甚至只要大概地了解就可以了。

第 16 章的内容对数据库管理员来说很有用。因为数据库系统的安全控制和用户管理是数据库管理员日常工作的一部分。对于其他的数据库人员，只要能基本上掌握本章的内容，甚至只要大概地了解就可以了。

如果您是一个数据库操作员（录入员），除了需要熟练掌握以上的内容之外，还需要熟练掌握第 12 章的内容。

如果您是一个数据库开发人员（程序员），除了需要熟练掌握以上的内容之外，还需要熟练掌握第 10 章、第 13 章和第 15 章的内容。

如果您是一个数据库管理员，除了需要掌握数据库操作员（录入员）应该熟练掌握的内容以外，还需要熟练掌握第 13 章、第 14 章和第 16 章的内容。

目　录

第0章

Oracle 的安装及相关配置

虽然本章的内容不是 Oracle 课程所必需的，但对读者进行上机操作和 Oracle Application Express（快速 Web 应用开发）的安装和配置十分必要。本章主要介绍如何在 Windows 系统上安装 Oracle 10g 或 Oracle 11g 以及相关工具的配置和使用。

0.1 Oracle的安装

安装 Oracle 之前，需要先安装 Windows 2000 Server、Windows Server 2003 或 Windows XP。

如果安装的是 Oracle 10g，则内存应该最少为 512MB，但是最好为 1GB 或以上。如果安装的是 Oracle 11g 并要安装 Oracle Application Express，内存最好为 2GB 或以上。

从 Oracle 体系结构或 SQL 的角度来看，从 Oracle 早期版本到 Oracle 最新的版本其变化很小。所以如果单纯是为了学习 Oracle SQL，安装 Oracle 10g 或 Oracle 11g 即可。如果读者的硬件比较紧张，也可以安装 Oracle 9i，甚至 Oracle 8 或 Oracle 8i（但是这两个版本不能在 Windows XP 上安装）。

在 Windows 操作系统上安装 Oracle 数据库管理系统并不太难，但要细心。其实在许多 Oracle 版本的安装过程中，除了 Oracle 系统的安装目录外，几乎不用做任何选择，都用默认值即可，甚至 Oracle 系统的安装目录也可以使用默认值。

提示：

随书的光盘中有 Oracle 10g 和 Oracle 11g 的安装视频，另外还包括了 Oracle 11g 的卸载视频。如果读者在安装 Oracle 系统时遇到问题可以参考光盘上的 Oracle 安装视频。

约定 1：如果没有特殊说明，本书的操作是在 Oracle 10g 或 Oracle 11g 版本上完成的。在遇到由于版本不同而引起的操作差别时，本书会加以说明。如果这些说明与所使用的系统无关，完全可以忽略它们。

约定 2：SQL 和 SQL*Plus 的语句是大小写无关的。尽管 Oracle 公司建议：“为了增加易读性，命令关键字一般为大写，而其他部分一般为小写”，但是实际情况并非如此。许多熟悉 UNIX 或 C 语言的用户倾向于整个语句全部小写，而许多熟悉 Windows 的用户又倾向于整个语句全部大写。为了使读者适应 Oracle 产业的这种实际情况，本书在使用 SQL

和 SQL*Plus 的语句时并不区分大小写。不过，建议读者在使用 SQL 或 PL/SQL 开发软件时，最好遵守 Oracle 公司的建议，这样会使软件的易读性增加，而且也更加易于维护。

约定 3：在如下命令方括号中的内容为可选项。下面的创建视图命令中[WITH READ ONLY]为可选项；竖线“|”为两者选一，如[FORCE|NOFORCE]；下划线为默认值，如NOFORCE。

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
AS subquery[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY];
```

在安装 Oracle 数据库管理系统之前，最好关闭防火墙之类的软件。以下是安装 Oracle 10.2.0.1.0（Oracle 10g）数据库管理系统的简化步骤（在安装之前可能需要先打补丁）：

（1）将 Oracle 10.2.0.1.0（Oracle 10g）数据库管理系统的第 1 张光盘插入光驱（如果没有选件，Oracle 8、Oracle 8i、Oracle 10g 或 Oracle 11g 只用 1 张光盘，但是 Oracle 9i 要使用 3 张光盘），Windows 操作系统会自动搜索 Oracle 系统的安装程序并运行该程序（如果 Windows 操作系统没有自动搜索到 Oracle 系统的安装程序，可以在光盘中找到 Setup 程序并运行它）。此时，会显示如图 0.1 所示的界面。

（2）在图 0.1 所示的界面中单击“开始安装”按钮，即会看到如图 0.2 所示的界面。

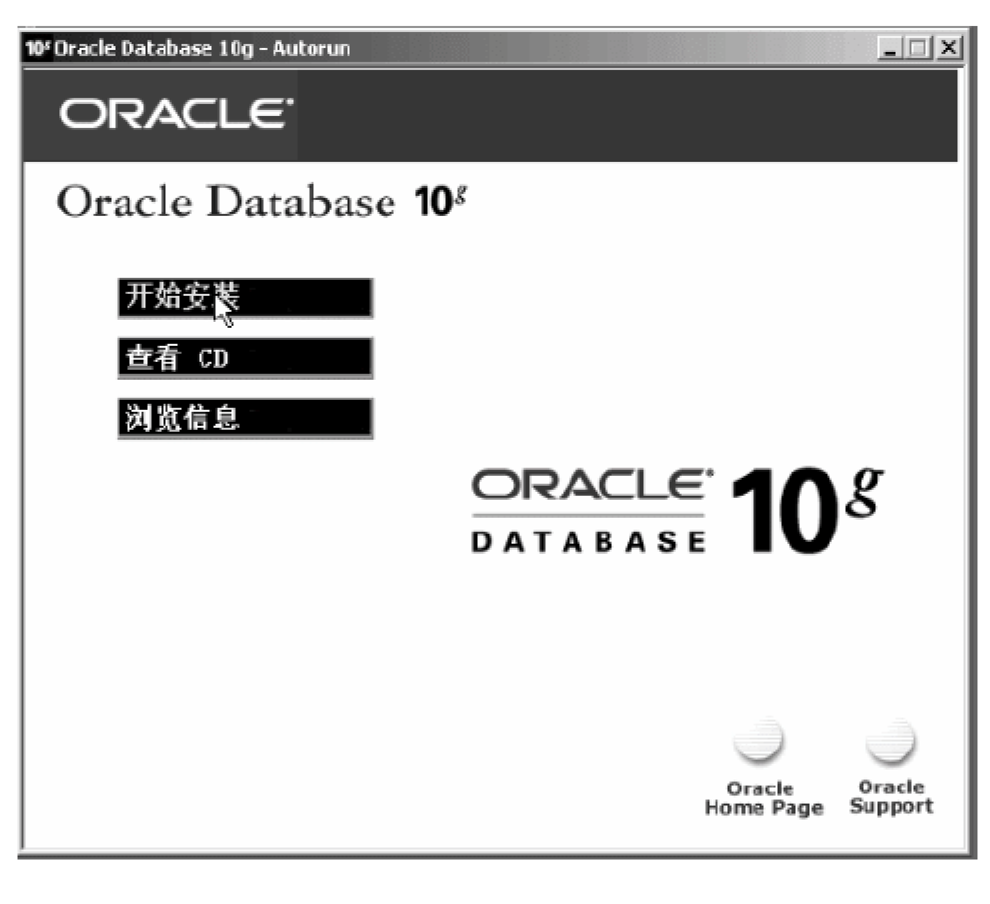


图 0.1

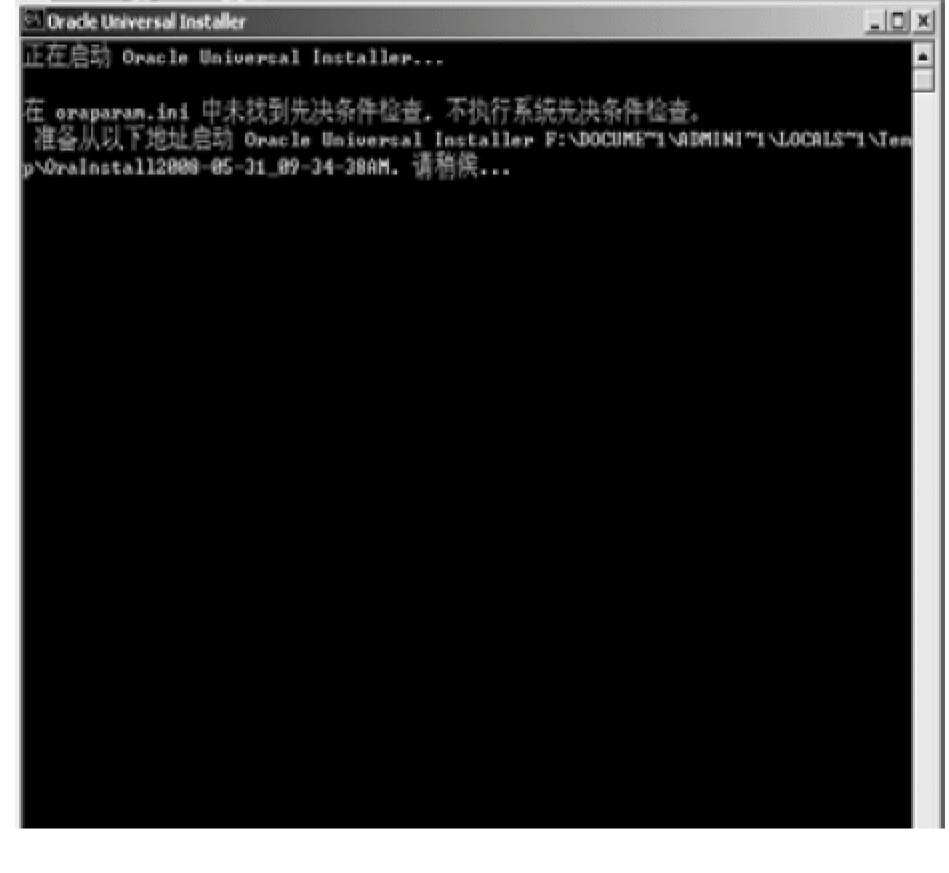


图 0.2

（3）图 0.2 表明，Oracle 此时正在检查操作系统的配置是否符合 Oracle 的安装要求，如果有问题就会报错，如果没问题就会进入如图 0.3 所示的界面。

（4）此时，可以修改 Oracle 安装目录和路径。例如，D 盘没有足够的磁盘空间，但 F 盘几乎是空的，就可以将路径改到 F 盘，也可以修改全局数据库名，还必须输入数据库的口令并确认。其中，数据库名和口令都是读者自己选的（可以选取任何您感兴趣的名，如数据库名为 dog，口令为 wangwang）。注意，在安装类型处应该选企业版，即为如图 0.4 所示的界面。

（5）单击“下一步”按钮，显示如图 0.5 所示的界面。图 0.5 为临时界面，当处理完

之后，会自动显示如图 0.6 所示的界面。



图　0.3



图　0.4



图　0.5

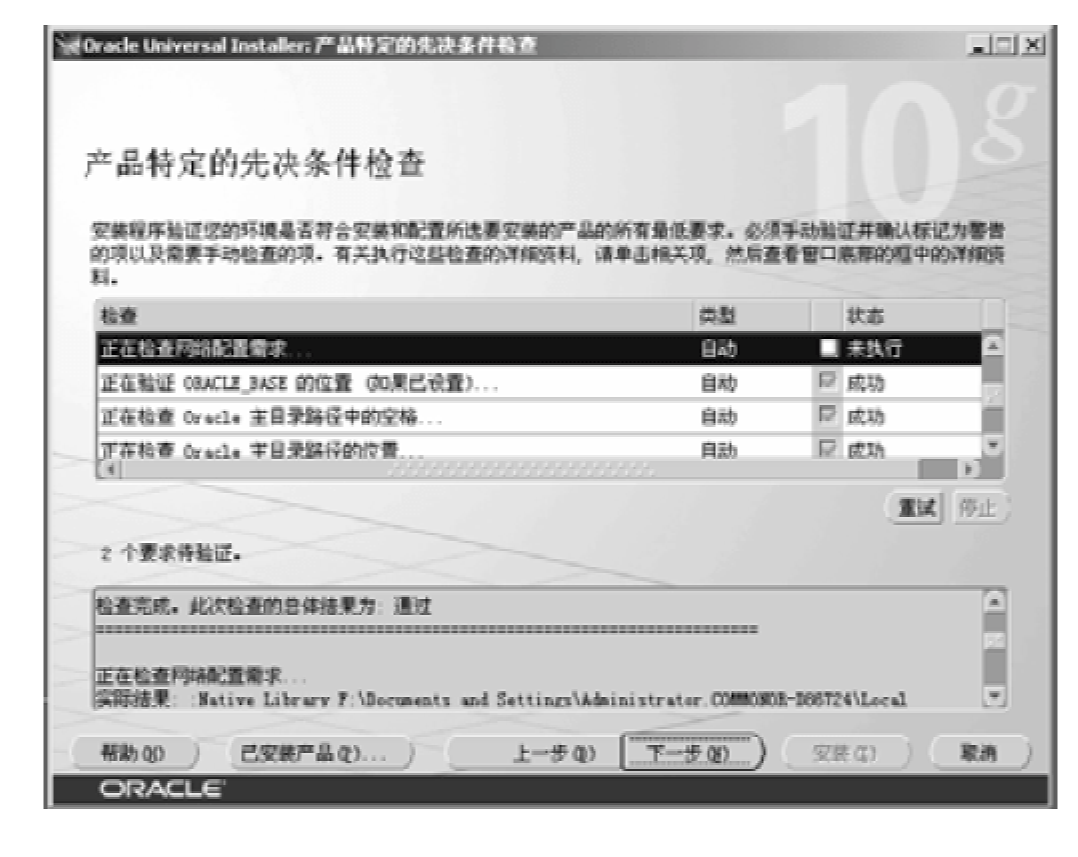


图　0.6

（6）此时，等待系统处理完之后，单击“下一步”按钮，显示如图 0.7 所示的界面。

（7）图 0.7 的界面是临时的，当处理进度达到 100%之后，即显示如图 0.8 所示的界面。

（8）图 0.8 的界面也是临时的，安装工作要进行一段时间。



图　0.7

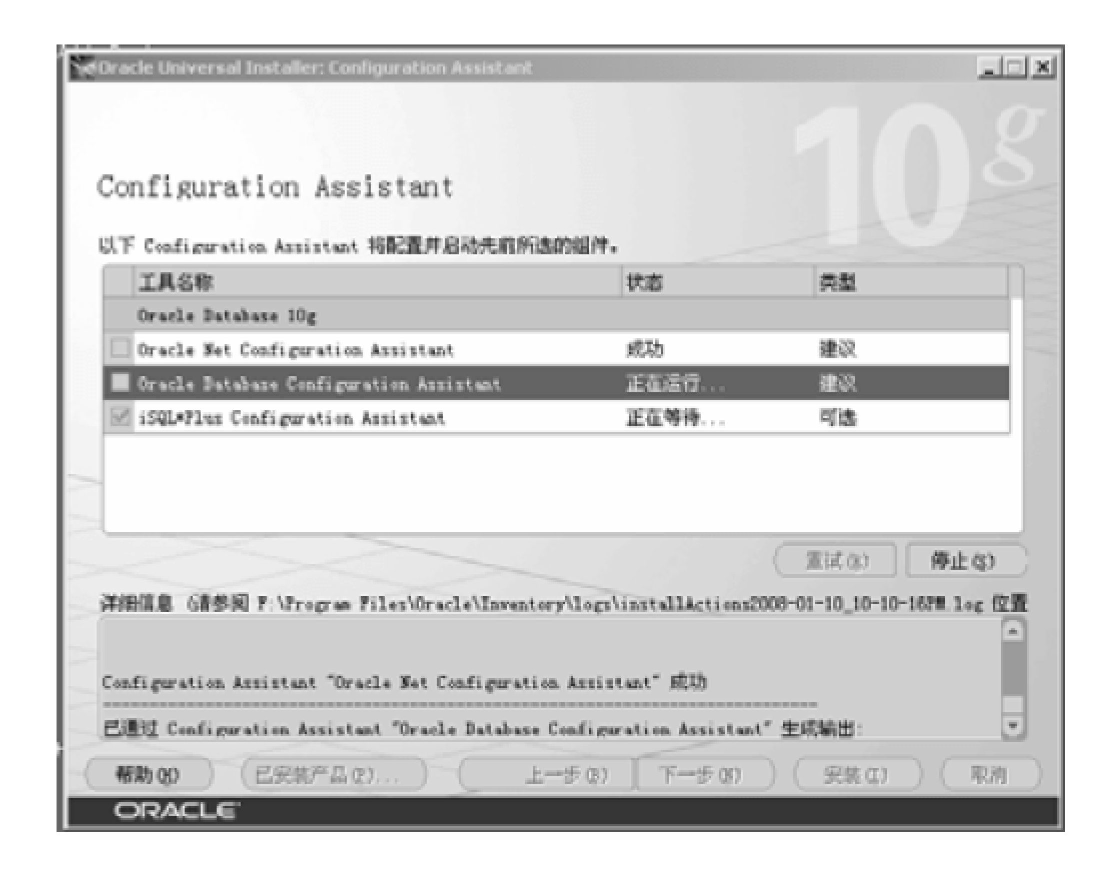


图　0.8

（9）此时，等待系统处理完之后，单击“下一步”按钮，显示如图 0.9 所示的界面。

（10）在图 0.9 所示的界面中可以单击“口令管理”按钮来修改 Oracle 默认用户的口令等。在第 1 次安装时，最好单击“确定”按钮，出现如图 0.10 所示的界面，表明 Oracle 数据库管理系统的安装已完成，此时可在图 0.10 中单击“退出”按钮。

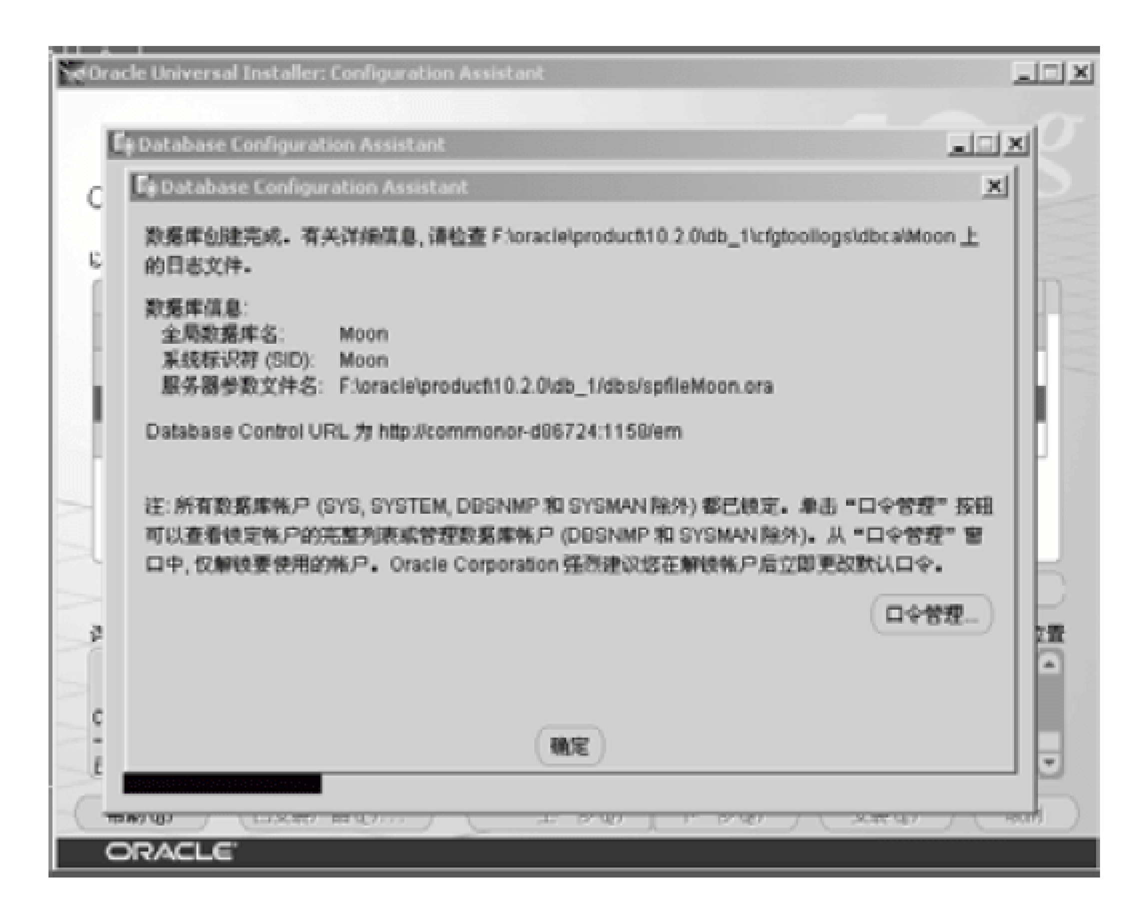


图 0.9



图 0.10

在实际安装 Oracle 时，一般系统都会提示输入数据库的名称，这时可以接受默认的数据库名，这个默认的数据库名与安装的 Oracle 的版本有关。如果安装的是 Oracle 8.17，其默认的数据库名为 ora817。另外，在 Oracle 9.2 或以上的版本中，在安装的过程中要求输入 sys 和 system 两个用户的口令。

提示：

第 1 次安装 Oracle 系统时，可以请人帮忙。因为一旦安装失败了，彻底卸载 Oracle 11g 之前的 Oracle 数据库系统并不是一件很容易的事。但是也用不着担心，只是需要多花些时间而已。最好的老师就是错误，每个人都能从错误中学到许多平时学不到的东西。

由于在 Oracle 10g 中必须使用 Internet 浏览器来登录 Oracle 10g 数据库企业管理器和 iSQL*Plus 图形工具，因此，在使用 Oracle 的图形工具之前，首先要获得它们的 HTTP 端口号(在 Oracle 11g 中登录企业管理器要使用 HTTPS 端口)，因此要进入$ORACLE_HOME\install 目录。其中，$ORACLE_HOME 为 Oracle 的安装目录（在笔者使用的这个电脑上为 F:\oracle\product\10.2.0\db_1\install），在这个目录下有一个叫 portlist.ini 的正文文件，在该文件中存放了所需的端口号，其中也包括了 iSQL*Plus HTTP 端口号。用户可以使用记事本打开这个文件。

iSQL*Plus 这个工具是 Oracle 9i 引入的，但是在 Oracle 9i 中其端口号存放在不同的文件中。iSQL*Plus 存放在$ORACLE_HOME\Apache\Apache\ports.ini 文件中，其中，$ORACLE_HOME 为 Oracle 的安装目录，如 E:\ORACLE\ora92\Apache\Apache\ports.ini 文件中。

需要指出的是，Oracle 11g 在默认安装时已经不再自动安装 iSQL*Plus 了，取而代之的是 Oracle SQL Developer 的图形开发工具，其功能更强大。

SQL*Plus 是一个重要的 Oracle 工具，它是所有 Oracle 版本必带的而且是自动安装的，利用它可以输入 SQL 语句，还可以进行 Oracle 数据库的管理与维护。下面简单介绍如何进入和使用 Oracle 的 SQL*Plus 界面。

0.2 进入Oracle的SQL*Plus界面

（1）选择如图 0.11 所示菜单中的命令，即可启动 Oracle 的 SQL*Plus 界面（为了以后操作方便，可以将 SQL*Plus 图标放到桌面上，其方法是：按下 Ctrl 键的同时用鼠标将其图标拖到桌面上）。

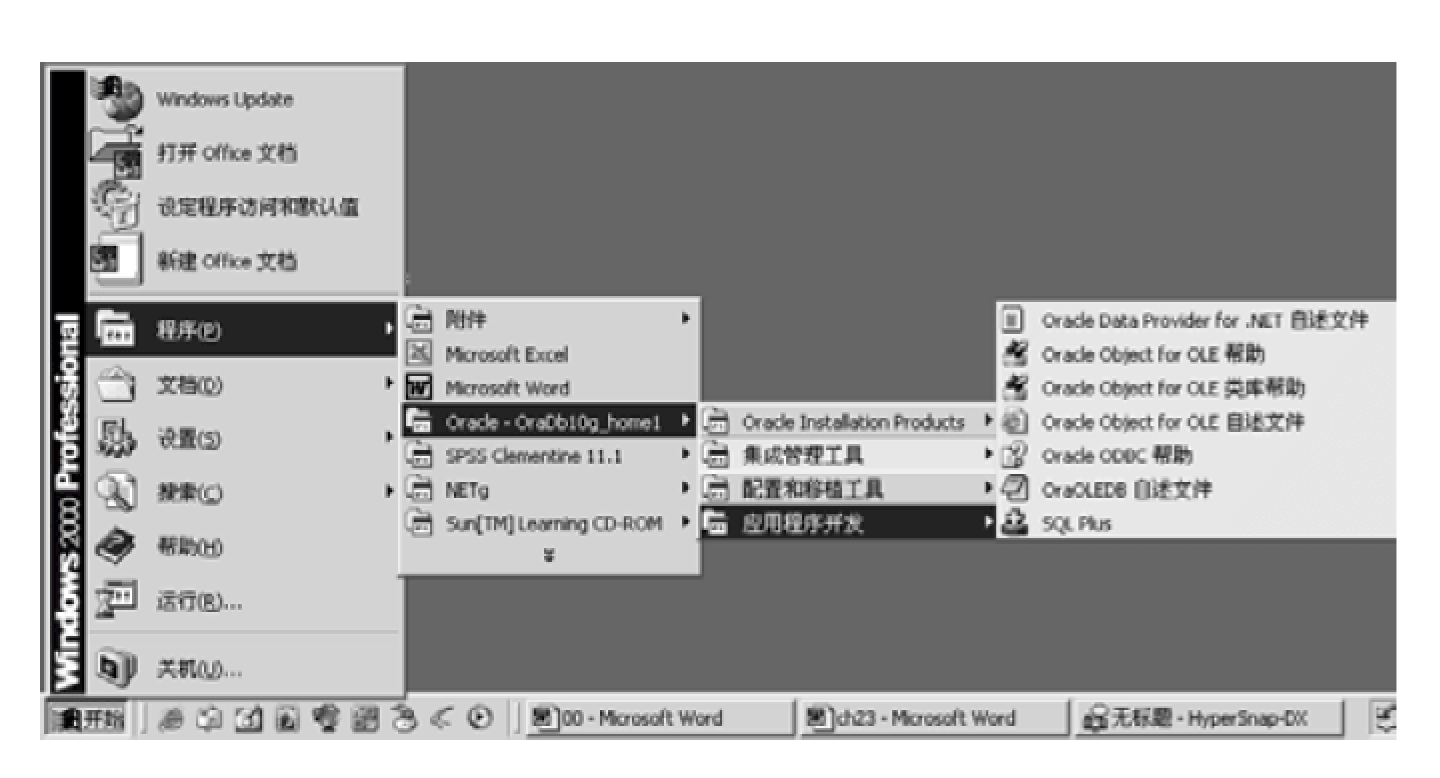


图 0.11

（2）在出现如图 0.12 所示的界面时，需要输入用户名和口令，Oracle 数据库中自动创建一个名为 scott 的用户，该用户的口令为 tiger（老虎），在这个用户账户中存有一些做练习所需的东西，如 emp（员工）表和 dept（部门）表。在“用户名”文本框中输入 scott，在“口令”文本框中输入 tiger，如图 0.13 所示。如果计算机上只有一个 Oracle 数据库或要连接的 Oracle 数据库为默认的数据库，就不必填写主机字符串，否则需要填写主机字符串。



图 0.12



图 0.13

（3）单击图 0.13 中的“确定”按钮，即出现如图 0.14 所示的 SQL*Plus 界面。此时，即可在“SQL>”提示符下输入 SQL 语句或 SQL*Plus 命令。

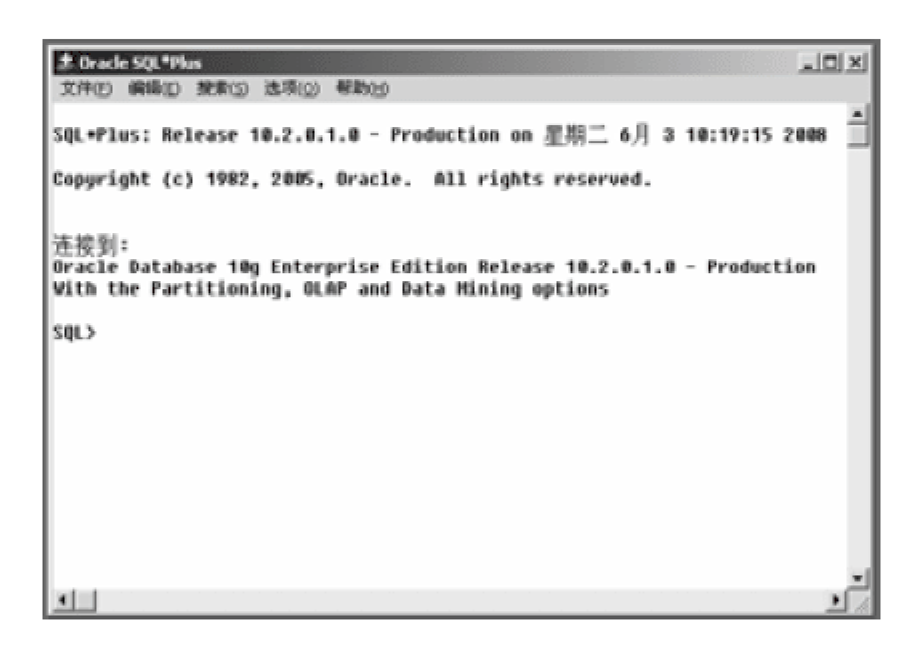


图 0.14

提示：

在 Oracle 10g 或以上的版本中，出于安全的考虑，所有 Oracle 的默认用户，包括 scott 用户都被锁住。此时，要先以 SYSTEM 或 SYS 用户登录数据库，即在图 0.13 中的“用户名”文本框中输入 system、在“口令”文本框中输入管理员口令（在安装 Oracle 数据库时输入的），之后使用如下命令：

```
alter user scott identified by tiger account unlock;
```

将 scott 用户的锁解开。

0.3　scott用户及其对象维护

在本书中，不少练习会用到 scott 用户中的表或其他对象。如果读者按本书的要求来做书中的例题，应该不会出现问题。但万一 scott 用户中的某个对象出现了问题该怎么办呢？也许有人会告诉您，要重装 Oracle 系统。如果真的碰上这样的人，相信过一会儿您就可以成为他的师傅了。

建议：

如果这种事情发生了，可以通过运行一个名为 scott.sql 的脚本文件来重建 scott 用户和它拥有的一切。在 Oracle 8i 或以上的版本中，这个脚本文件在$ORACLE_HOME\rdbms\admin 目录下。$ORACLE_HOME 是指 ORACLE 系统的安装目录。

在笔者的计算机上一个 Oracle 10g 数据库系统的$ORACLE_HOME （Oracle 安装目录）为 F:\oracle\product\10.2.0\db_1，所以该脚本文件的路径和名称为 F:\oracle\product\10.2.0\db_1\RDBMS\ADMIN\scott.sql。

当以数据库管理员用户 system 登录系统之后，在 SQL>提示符下运行该脚本文件，命令如下：

```
SQL> @F:\oracle\product\10.2.0\db_1\RDBMS\ADMIN \scott.sql
```

之后，Oracle 系统将重新安装 scott 用户和该用户下的所有表和其他对象。

0.4　本书中将用到的表

读者在 SQL 的学习中将使用的表主要有 3 个，它们都属于用户 scott，分别为存放员工详细信息的员工表 emp、存放部门信息的表 dept 及存放工资和工种级别的表 salgrade，如图 0.15、图 0.16 和图 0.17 所示。

emp 表



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12月-80	800		20
7499	ALLEN	SALESMAN	7698	20-2月 -81	1600	300	30
7521	WARD	SALESMAN	7698	22-2月 -81	1250	500	30
7566	JONES	MANAGER	7839	02-4月 -81	2975		20
7654	MARTIN	SALESMAN	7698	28-9月 -81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-5月 -81	2850		30
7782	CLARK	MANAGER	7839	09-6月 -81	2450		10
7788	SCOTT	ANALYST	7566	19-4月 -87	3000		20
7839	KING	PRESIDENT		17-11月-81	5000		10
7844	TURNER	SALESMAN	7698	08-9月 -81	1500	0	30
7876	ADAMS	CLERK	7788	23-5月 -87	1100		20
7900	JAMES	CLERK	7698	03-12月-81	950		30
7902	FORD	ANALYST	7566	03-12月-81	3000		20
7934	MILLER	CLERK	7782	23-1月 -82	1300		10

图　0.15

dept 表

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

图　0.16

salgrade 表

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

图　0.17

0.5　SQL（Structured Query Language）语言

本书的前 17 章将全面系统地介绍 SQL 语言。SQL 语言包括的内容如下：

- 数据查询语言。
- 数据操作（维护）语言（Data Manipulation Language——DML）。
- 数据定义语言（Data Definition Language——DDL）。
- 事务控制（Transaction Control）。
- 数据控制语言（Data Control Language——DCL）。

数据查询语言包括 SELECT 语句（有些 Oracle 书籍将其归入数据操作语言）。

数据操作（维护）语言包括 INSERT、UPDATE 和 DELETE 语句。

数据定义语言包括 CREATE、ALTER、TRUNCATE、RENAME 和 DROP 语句。

事务控制包括 COMMIT 和 ROLLBACK 语句。

数据控制语言包括 GRANT 和 REVOKE 语句。

本书的第 1~17 章将全面系统地介绍以上所有的语句。同时，为了开发和管理的需要，还将详细介绍 SQL*Plus 的环境参数的配置和替代变量的定义等。

0.6 本书所用的术语

下面简单介绍在本书中使用的一些数据库和计算机方面的术语。为了解释方便，利用图 0.18 给出一些数据库术语的图形化说明。

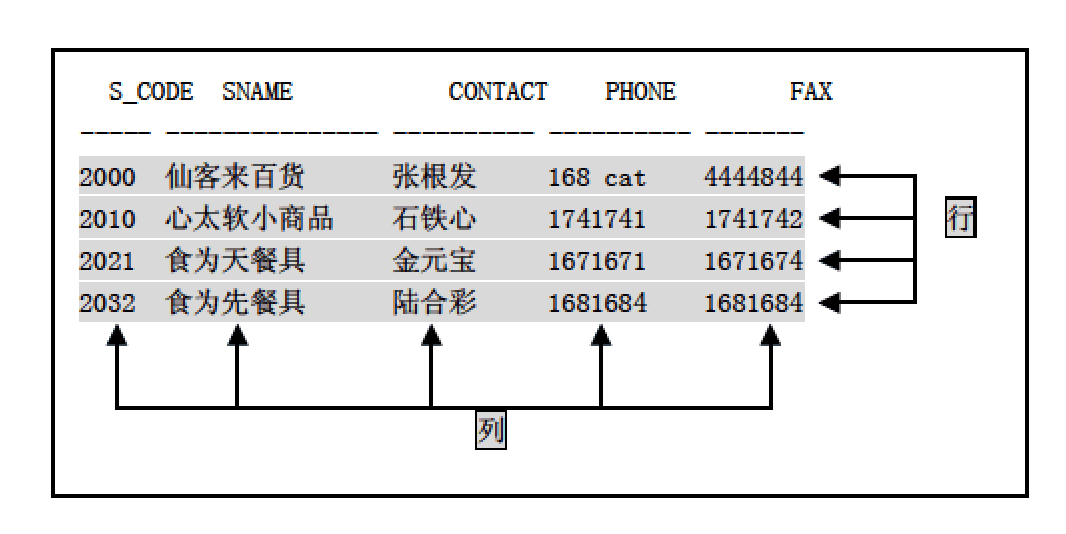


图 0.18

由图 0.18 给出如下的术语和它们的定义。

- 表（table）：是由行和列组成的二维结构。
- 行（row）：每一行给出了一个供应商的全部信息（记录）。
- 列（column）：每一列表示供应商的一种特性（属性）。
- 值（value）：行和列的交汇处。如第 2 行和第 3 列的交汇处为“石铁心”，即表示第 2 行的联系人（CONTACT）为“石铁心”。

为了减少初学者学习的难度，本书并没有给出学术术语的严格定义，也没有很严格地区别一些术语。在阅读本书时注意如下的约定：

表（table）=实体（entity）=关系（relation）

行（row）=记录（record）

列（column）=属性（attribute）

ORACLE 服务器（SERVER）= ORACLE 系统= ORACLE 数据库（管理）系统

没有阴影的内容是用户要输入的，但不包括“SQL>”和行号。如在下例中只需输入 SELECT*和 FROM supplier;。“SQL>”为 ORACLE SQL*Plus 的提示符；2 是 SQL 语句的行号，由 SQL*Plus 自动产生。

```
SQL> SELECT *
2  FROM supplier;
```

阴影部分为 SQL*Plus 产生的 SQL 语句的显示输出。下面就是以上所用到的 SQL 语句产生的输出结果：

```
S_CODE  SNAME              CONTACT      PHONE       FAX
------- ----------------   ------------ ----------  --------
   2000 仙客来百货          张根发       168 cat     4444844
   2010 心太软小商品        石铁心       1741741     1741742
   2021 食为天餐具          金元宝       1671671     1671674
   2032 食为先餐具          陆合彩       1681684     1681684
```

本书中绝大部分的例子都是在 Oracle SQL*Plus 上完成的。除了 SQL*Plus 之外，还有很多其他的工具可以用作输入和运行 SQL 语句，例如，Oracle 9i 引入的 iSQL*Plus 和 Oracle 10g 开始引入的 SQL Developer。到了 Oracle 11g，SQL Developer 已经成为默认安装的工具。

之所以主要使用 SQL*Plus，是因为它在 Oracle 的所有版本中都能得到。只要安装了 Oracle 数据库（管理）系统，上面就一定有 SQL*Plus。这样如果学会了使用 SQL*Plus 等于有了一个“看家”的本领。即当您遇到实际的 Oracle 系统时，无论它运行在什么 IT 平台上，无论它使用的是什么工具，都可以立即开始工作。

要想在无情的商海中生存，能够立即开始工作这一点有时是很重要的。特别是刚刚找到一份新工作的用户，或者作为一名 Oracle 顾问到现场为客户解决实际问题而对客户的 Oracle 系统的配置又一无所知时，SQL*Plus 就非常有用。另外，其他的命令行工具与 SQL*Plus 的差别很小，一般情况下，用户用很短的时间就可以适应。

0.7　Oracle 11g上的SQL*Plus

Oracle 11g 默认安装的 SQL*Plus 并不是在 0.2 节介绍的 Oracle 公司称之为图形界面的 SQL*Plus，而是一种命令行界面的 SQL*Plus。在 Oracle 11g 上，启动 SQL*Plus 的具体操作步骤如下：

（1）单击“开始”按钮，选择“所有程序”→Oracle-OraDb11g_home1→“应用程序开发”→SQL Plus 命令，如图 0.19 所示。

（2）在“Enter user-name:”提示符后处输入 scott，在“Enter password:”提示符后输入 tiger，如图 0.20 所示。




图　0.19

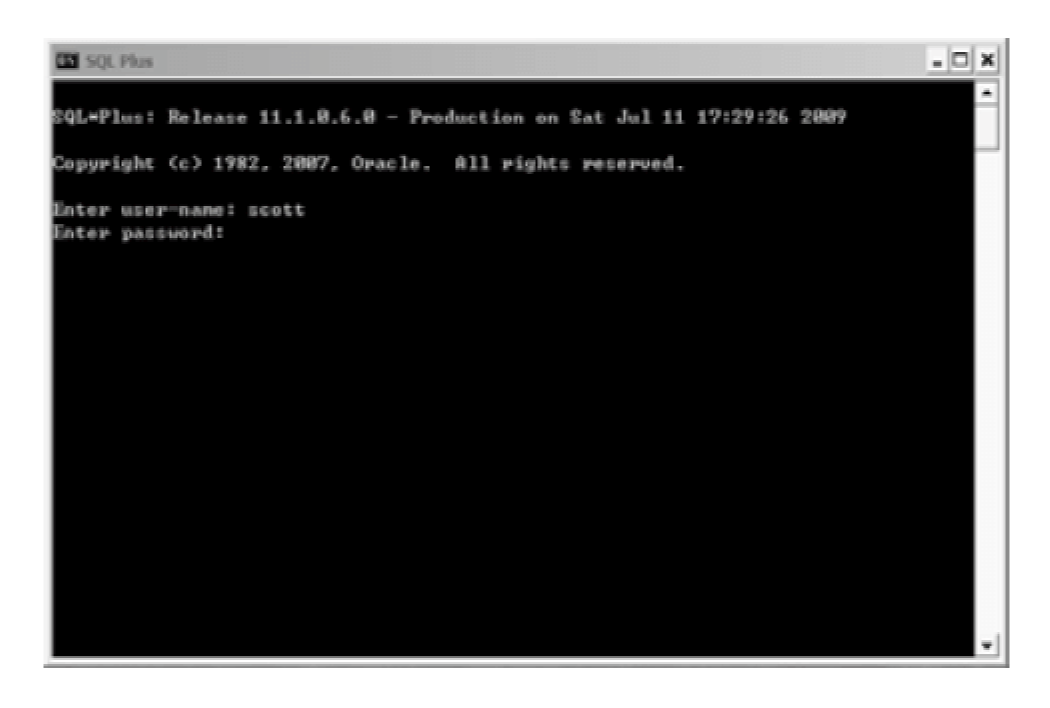


图　0.20

（3）按 Enter 键即可启动 SQL*Plus 并以 scott 用户登录 Oracle 数据库，如图 0.21 所示。

（4）此时即可输入 SQL 语句或 SQL*Plus 命令，如图 0.22 所示。



图 0.21

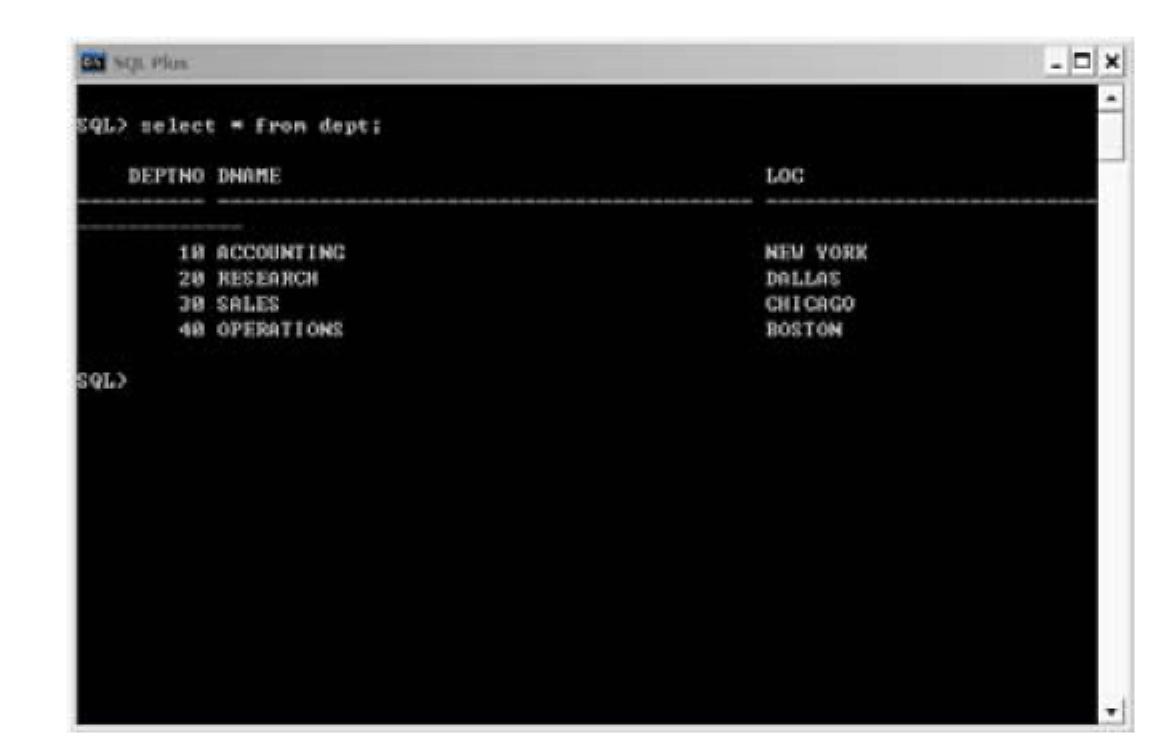


图 0.22

0.8 使用iSQL*Plus

从 Oracle 9i 和 Oracle 10g 开始，Oracle 还提供了另一个工具 iSQL*Plus，它是网络版的 SQL*Plus，通过 Internet 浏览器登录。因此，它需要首先获得 iSQL*Plus 服务的 HTTP 端口号。因此要进入$ORACLE_HOME\install 目录（在笔者的电脑上为 E:\oracle\product\10.2.0\db_1\install；在您的系统中 Oracle 可能安装在其他盘上），在这个目录下有一个叫 portlist.ini 的正文文件，如图 0.23 所示。

选择文件 portlist.ini 并双击将该文件打开（使用记事本打开）。在该文件中存有 iSQL*Plus 的 HTTP 端口号，为 5560，如图 0.24 所示。

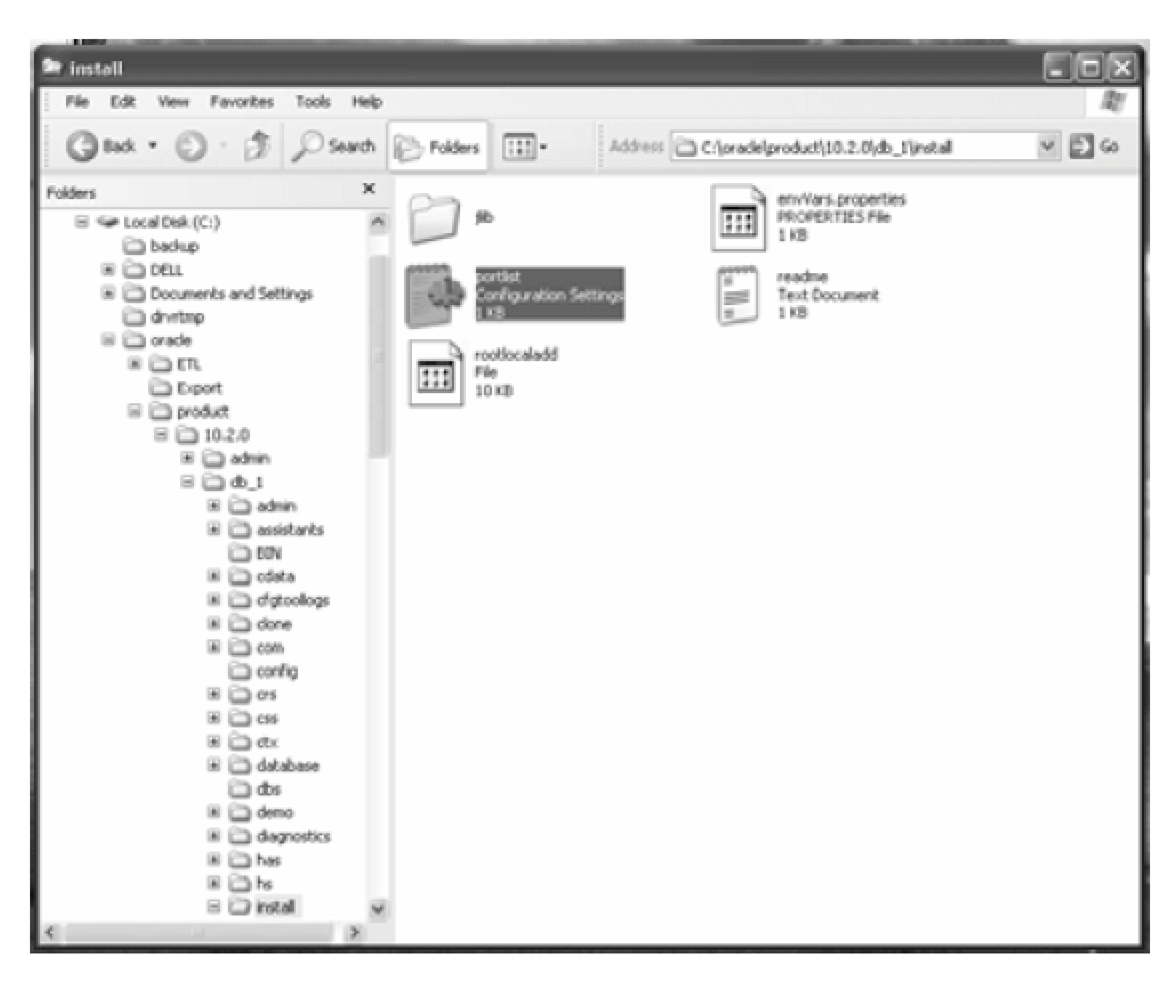


图 0.23

Enterprise Manager 代理端口 (jinlian) = 3938
Enterprise Manager Console HTTP 端口 (jinlian) = 1158
iSQL*Plus HTTP 端口号 =5560

图 0.24

当获得了 iSQL*Plus 的 HTTP 端口号之后，就可以使用网络浏览器利用 iSQL*Plus 来登录 Oracle 数据库。现在启动 Internet 浏览器，并在 Internet 浏览器中输入 http://localhost:

5560/isqlplus（如果是远程登录，则要将 localhost 换成主机名或 IP 地址）。如果一切正常，则应该出现 isqlplus 的登录界面。但是也有可能看到如图 0.25 所示的出错界面，这是因为 isqlplus 服务（在 UNIX 和 Linux 系统上是进程）没有启动。为了看上去更专业，下面使用命令行的方式启动 isqlplus 进程（在 UNIX 和 Linux 系统上一般只能使用这种方式）：

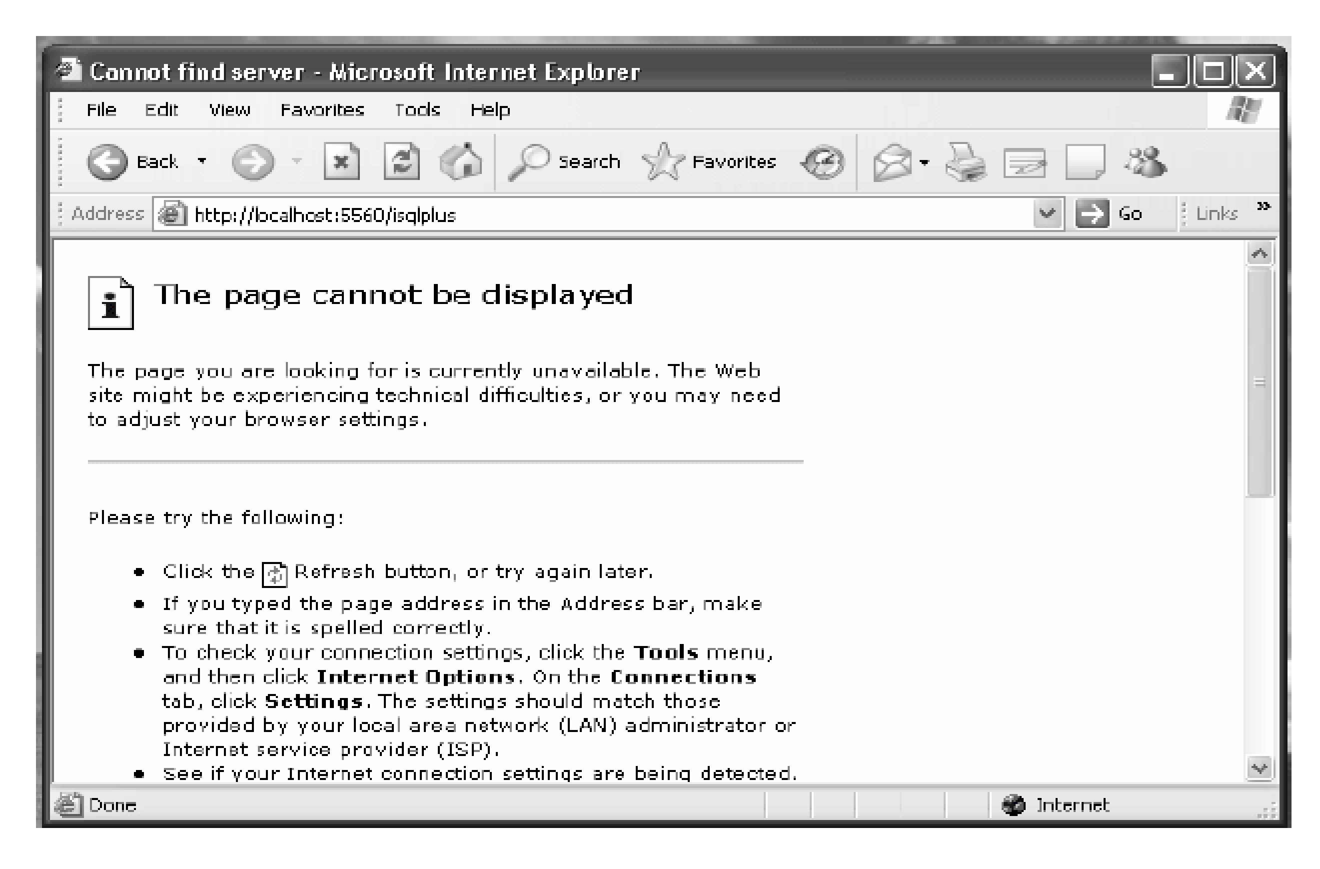


图　0.25

（1）选择“开始”→“所有程序”→“附件”→“命令提示符”命令，启动 DOS 窗口，如图 0.26 所示。

（2）使用 isqlplusctl start 的命令启动 isqlplus 进程（如果您的系统环境变量没有设好，需要切换到 isqlplusctl 应用程序所在目录，如 E:\oracle\product\10.2.0\db_1\BIN），如图 0.27 所示。



图　0.26



图　0.27

（3）然后再次启动 Internet 浏览器，并在 Internet 浏览器中重新输入 http://localhost:5560/isqlplus/，这次就会出现 isqlplus 的登录界面，如图 0.28 所示。

（4）此时，输入用户名和密码后单击“登录”按钮即可登录 Oracle 数据库系统，如图 0.29 所示。

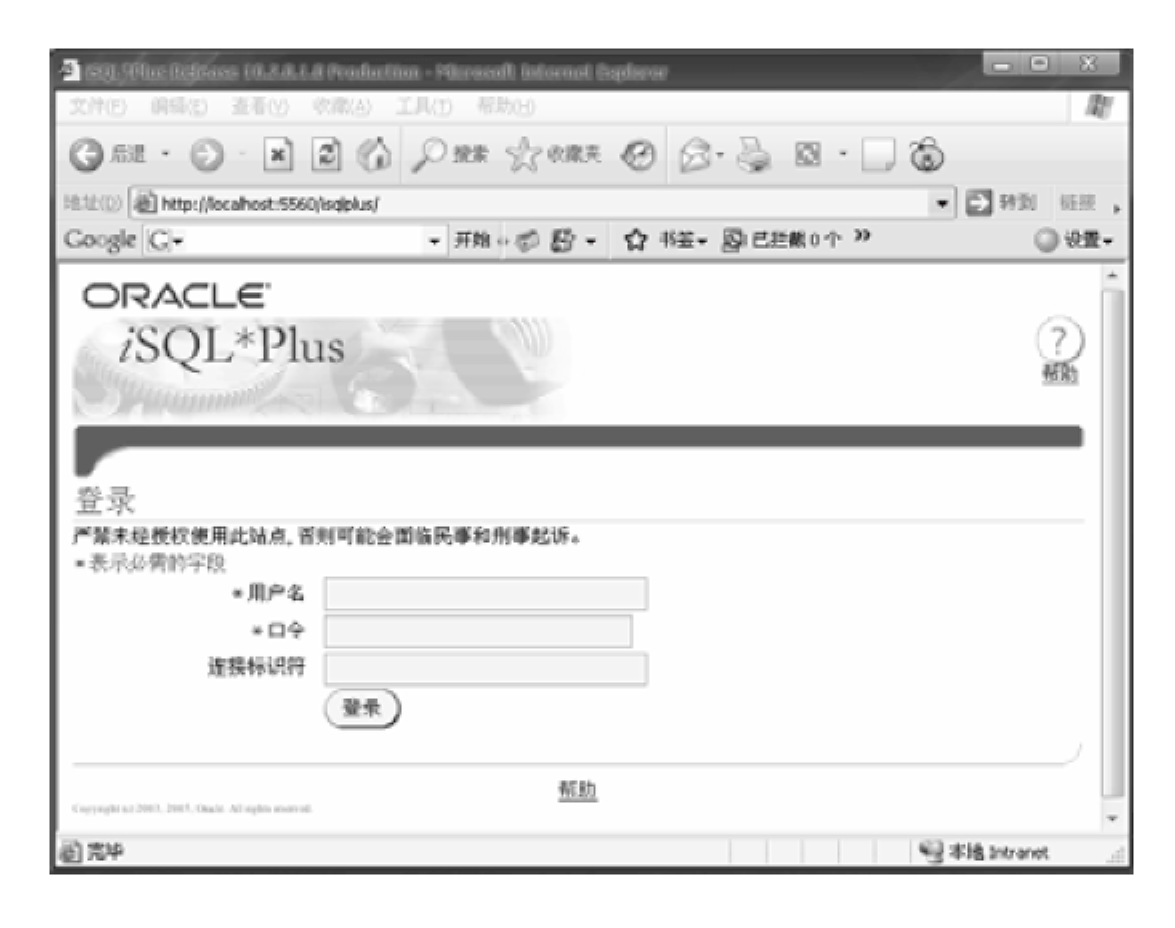


图　0.28

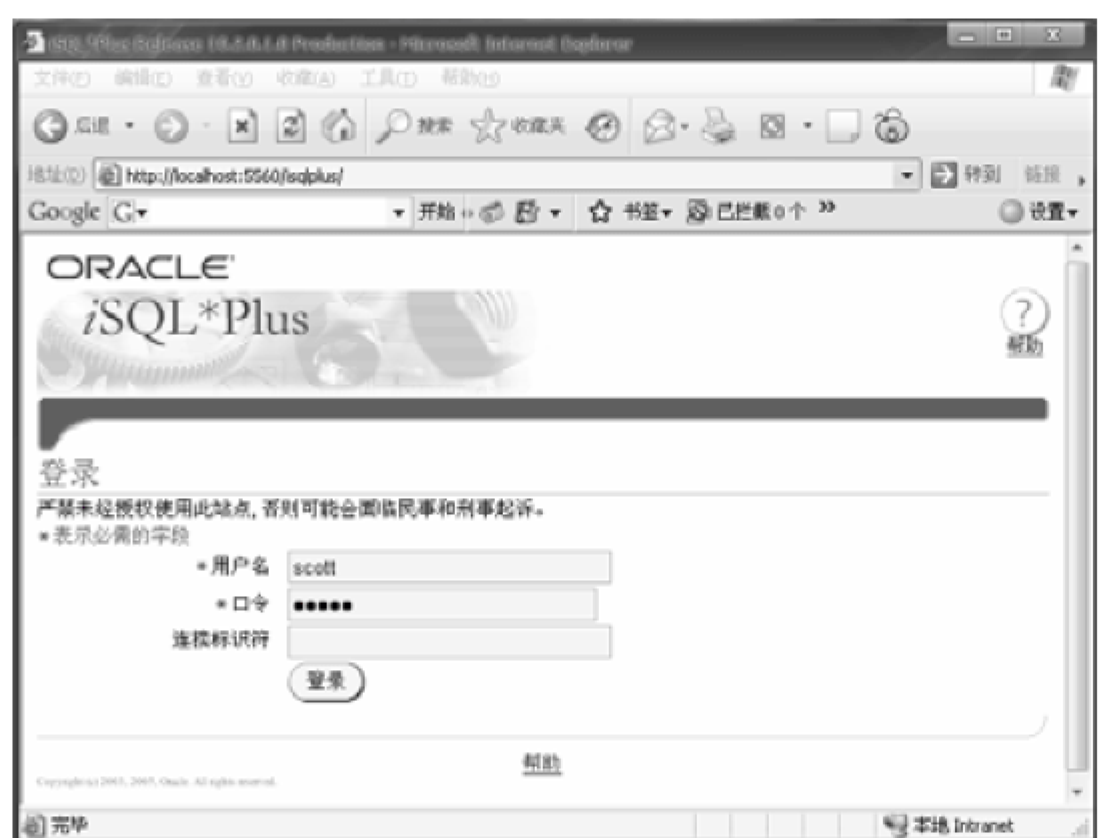


图　0.29

（5）在工作区中输入 SQL 语句 select * from emp 后单击“执行”按钮执行这一 SQL 查询语句，如图 0.30 所示。



图　0.30

（6）然后就会得到这个 SQL 查询语句的结果，向下滚动最左边的滚动条来查看所得的查询结果，如图 0.31 所示。



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12月-80	800		20
7499	ALLEN	SALESMAN	7698	20-2月 -81	1600	300	30
7521	WARD	SALESMAN	7698	22-2月 -81	1250	500	30
7566	JONES	MANAGER	7839	02-4月 -81	2975		20
7654	MARTIN	SALESMAN	7698	28-9月 -81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-5月 -81	2850		30
7782	CLARK	MANAGER	7839	09-6月 -81	2450		10
7788	SCOTT	ANALYST	7566	19-4月 -87	3000		20
7839	KING	PRESIDENT		17-11月-81	5000		10
7844	TURNER	SALESMAN	7698	08-9月 -81	1500	0	30
7876	ADAMS	CLERK	7788	23-5月 -87	1100		20
7900	JAMES	CLERK	7698	03-12月-81	950		30
7902	FORD	ANALYST	7566	03-12月-81	3000		20
7934	MILLER	CLERK	7782	23-1月 -82	1300		10

已选择14行。

图　0.31

提示：

isqlplus 并不是 Oracle 10g 才引入的，该工具在 Oracle 9i 就引入了。但是在 Oracle 9i 中，它的端口号存在于不同的文件中。isqlplus 存在$ORACLE_HOME\Apache\Apache\ports.ini 文件中，如 E:\ORACLE\ora92\Apache\Apache\ports.ini 文件中。在 Oracle 11g 中默认并不安装 isqlplus。

0.9　使用DOS窗口启动SQL*Plus

在所有的 Oracle 版本上，读者都可以在 DOS 命令行下启动 SQL*Plus，其具体操作步骤如下：

（1）在 Windows 操作系统上启动 DOS 窗口（命令行窗口），在命令行提示符下输入命令 sqlplus scott/tiger 后按 Enter 键，如图 0.32 所示。

（2）使用上面的方法启动 SQL*Plus 虽然方便但是存在安全隐患，因为其他人可以看到您的密码。另一种安全启动 SQL*Plus 的方法是在启动 SQL*Plus 时只输入用户名，然后在提示处输入密码，如图 0.33 所示。

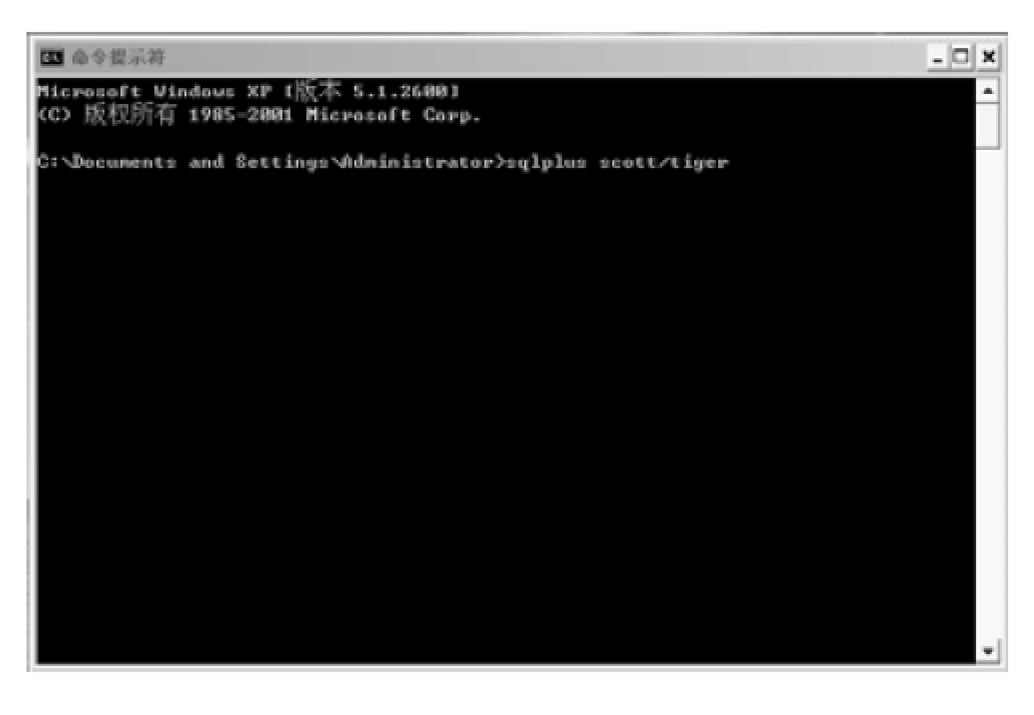


图　0.32

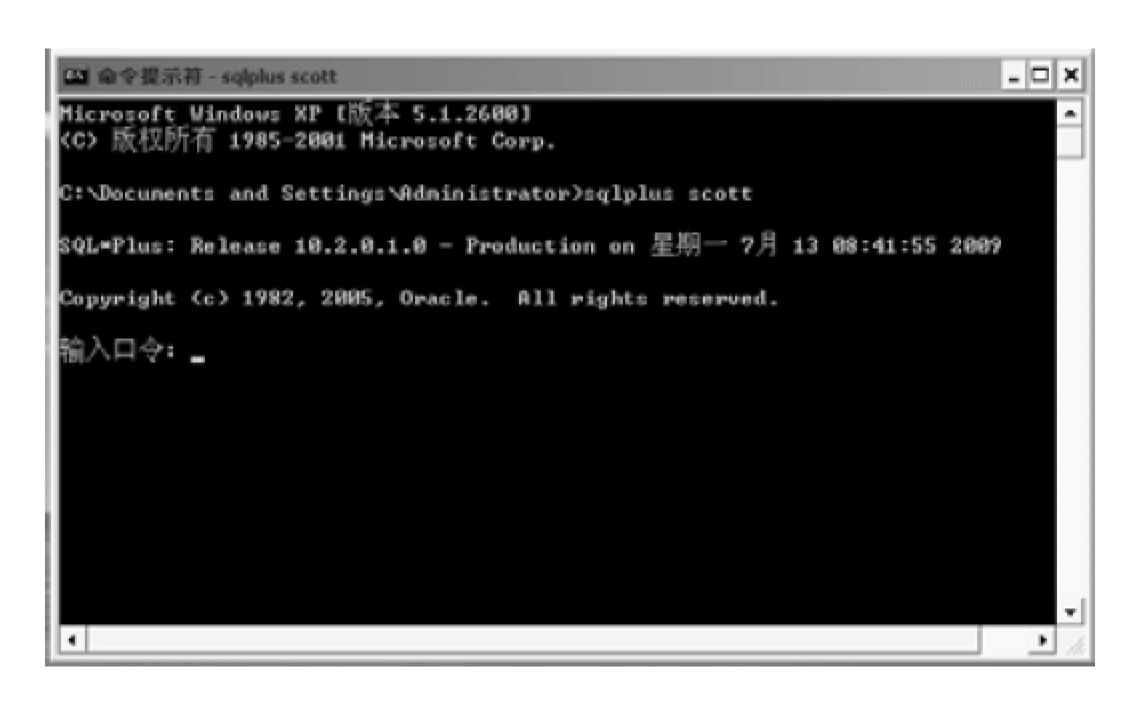


图　0.33

（3）进入 SQL*Plus 后就可以输入 SQL 语句了，如输入 SQL 查询语句 select*from dept 并按 Enter 键运行这一语句，如图 0.34 所示。

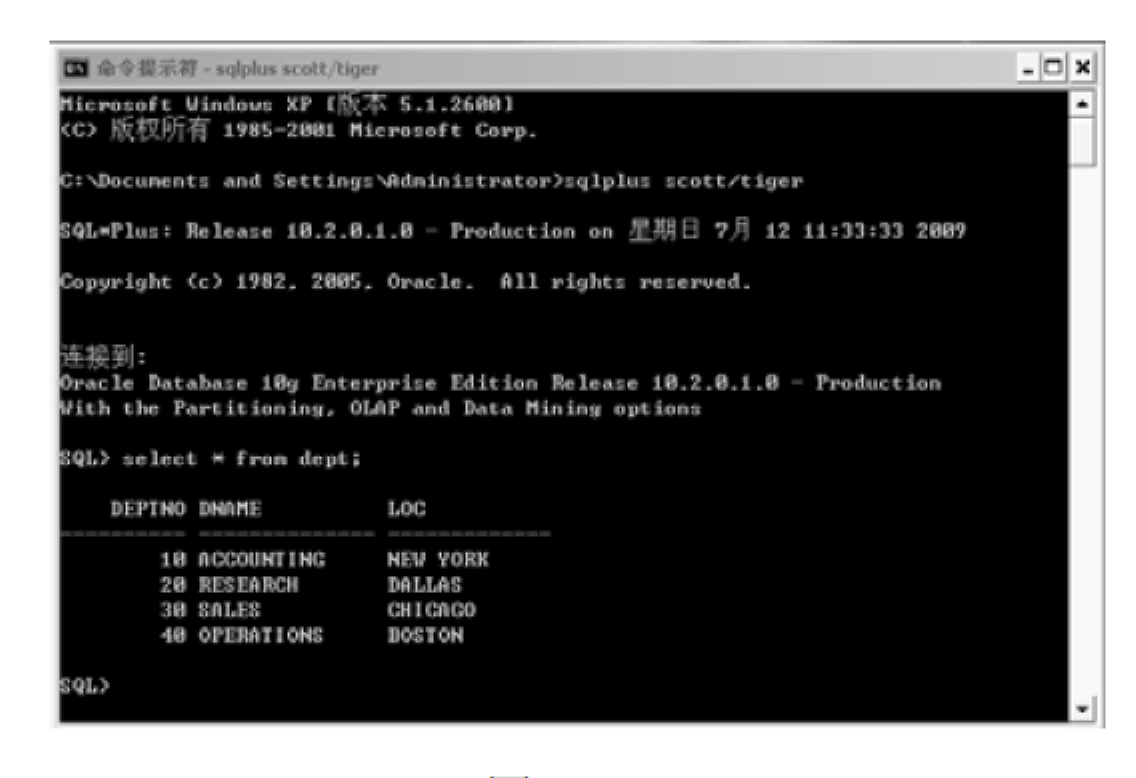


图　0.34

在这种 SQL*Plus 中，读者可以使用键盘上的上箭头找到之前执行的命令，然后加以修改并运行，读者也可以利用上、下箭头在使用过的命令之间移动。许多 Oracle 的“大虾”喜欢使用这种 SQL*Plus，特别是在用户面前。读者知道为什么吗？

有人认为是操作方便，其实图形方式的 SQL*Plus 也很方便；也有人认为是保护眼睛，因为黑屏辐射小，其实答案是：Looks very professional（看上去非常专业）。因为许多用户根本就没使用过命令行界面，一看到就觉得眼晕。生活当中也是一样，现在流行的一句话是“喜欢的歌静静地听，喜欢的人远远地看”，因为远远地看就看不清，看不清的就美，就是所谓的朦胧美。所以有专家建议，如果真心爱上了一个人，千万别和他/她结婚，因为这样在一生中都会有一个不醒的美梦，这将成为您人生旅途中最后避风的港湾。境界高吧？但是有更高的境界，就是在虚拟世界中的网恋，根本就看不到对方。从现在起您也可以在其他用户面前朦胧起来了，这样很快在其他用户眼里您就成了“大虾”，甚至“泰斗”或“宗师”。

第1章 简单查询语句

SQL（Structured Query Language）是标准的关系数据库（Relational Database Management Systems）操作语言。SQL 语言包括查询语言（Data Retrieval）、数据操作语言（Data Manipulation Language，DML）、数据定义语言（Data Definition Language，DDL）、数据控制语言（Data Control Language，DCL）和事务控制（Transaction Control）。SQL 是一种非过程化的第四代高级语言，它的语法与英语非常相似，因此它是一种很容易学习的计算机语言。对初学者几乎没有任何要求，换句话说，初学者可以没有任何计算机背景。

本书将详细地介绍以上各个部分的内容。为了讲解方便，在本书中将利用一个虚构的跨国公司——Sun & Moon Limited Corporation（日月神有限公司）来介绍 SQL 语言在实际商业环境中的应用。以下是日月神有限公司的简介。

日月神有限公司是一个总部设立在一个白云缭绕的太平洋岛国上的大型跨国公司。该公司的总部原来设在某个西方国家的首都。日月神有限公司的创始人，也是该公司的老板，是一个难得的企业家，公司在他的领导下生意蒸蒸日上，已经成为一个实力雄厚的大型跨国公司，其经营的范围涵盖了许多领域，该公司的触角已遍及五大洲，并在许多国家设立了子公司。

该公司在很早就开始利用 Oracle 数据库管理系统建立了信息系统，用来减轻公司管理人员的日常工作负担和帮助公司的经理们作出正确的决策。但是日月神的老板并不认为信息技术（IT）能决定公司的命运。他一直认为真正决定公司成败的是人不是技术。

正是基于“真正决定公司成败的是人不是技术”这一理念，日月神公司在它的信息管理系统中最先实现的是人事管理部分。这部分包含了 3 个最重要的表，分别为员工表（emp）、部门表（dept）和工资级别表（salgrade）。其中：

- 员工表（emp）包含了日月神公司中所有员工的基本信息，如姓名和工资等。
- 部门表（dept）包含了日月神公司中所有部门的基本信息，如部门名和所在地址等。
- 工资级别表（salgrade）包含了每一工资级别的工资下限和上限。

本书将主要利用这 3 个表讲解 SQL 语言和 SQL*Plus 命令。虽然日月神公司有几十万甚至上百万名员工和几百个或上千个部门，但为了讲解方便，我们只使用这些表中非常小的一部分。

在本书中有如下的约定：如果没有明确说明，书中所介绍的一切都发生在日月神公司；书中的老板、老总、总裁等就是指日月神的老板；书中的您是指日月神公司的数据库管理员（DBA）或数据库操作员。现在我们就从最简单的查询语句开始 SQL 语言学习的愉快旅程。

1.1　最简单的查询语句

在查询语句中实际上只有 SELECT 和 FROM 子句是必需的，也是最简单的查询语句。如果您的经理想知道公司中所有员工的信息，可以使用例 1-1 的查询语句来得到他所需要的信息。假设公司所有员工的信息都保存在一个叫 emp（员工）的表中。

例 1-1

```
SQL> SELECT *
  2  FROM emp;
```

例 1-1 结果

```
     EMPNO  ENAME                 JOB                             MGR  HIREDATE
----------  --------------  ------------------------------  ---------
      SAL        COMM       DEPTNO
---------  ---------  ------------
     7369  SMITH                  CLERK                           7902  17-12? -80
      800                       20

     7499  ALLEN                  SALESMAN                        7698  20-2?  -81
     1600          300          30

     7521  WARD                   SALESMAN                        7698  22-2?  -81
       1250          500          30    …
```

注意：

在本书中采用如下的约定：

- 没有阴影的内容为输入的 SQL 语句或 SQL*Plus 命令。
- 阴影中的内容为系统显示（输出）的结果。

这里的“*”号表示所有的列，它与在 SELECT 之后列出所有的列名是一样的。查询语句以分号（;）结束。emp（employee）是一个员工表。显示结果后的（…）表示在其后还有显示被省略了。

Oracle 有许多 UNIX 和 C 的影子，在对表和列命名时大量地使用缩写形式，如 salary（工资）——sal。可能您已经注意到了 HIREDATE 一列的显示有些奇怪，这是因为 emp 表中存的是英文数据，而我使用的是中文操作系统。以后会再介绍转换的方法。

1.2　在查询语句中如何选择特定的列

您也许觉得上面的显示有些乱。在许多情况下，我们只想知道若干个列的信息，使用 SELECT 语句很容易做到。在现实生活当中也是这样，如您到超市去买东西，当然不会买

下超市中的所有东西。例如：

买 羊肉，兔肉
从 大灰狼超市;

又如公司的会计在每次发工资时，她可能需要确定每个员工的工号（empno）、名字（ename）和工资（sal）。可以使用例 1-2 的查询语句来实现她的这一要求。

例 1-2

```
SQL> SELECT empno, ename, sal
  2  FROM emp;
```

例 1-2 结果

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7369 SMITH             800
      7499 ALLEN            1600
      7521 WARD             1250
      7566 JONES            2975
      7654 MARTIN           1250
      7698 BLAKE            2850
      7782 CLARK            2450
      7788 SCOTT            3000
      7839 KING             5000
      7844 TURNER           1500
      7876 ADAMS            1100
      7900 JAMES             950
      7902 FORD             3000
      7934 MILLER           1300
已选择 14 行。
```

在查询语句中选择指定的列就是关系数据库中所称的投影（Project）。可以在 SELECT 之后选择任意列，列与列之间以逗号隔开，而且可以根据需要来指定列的显示次序，如例 1-3 所示。

例 1-3

```
SQL> SELECT sal,ename,empno
    2  FROM emp;
```

例 1-3 结果

```
       SAL ENAME                     EMPNO
---------- -------------------- ----------
       800 SMITH                      7369
      1600 ALLEN                      7499
      1250 WARD                       7521
```

```
      2975 JONES                     7566
      1250 MARTIN                    7654
      2850 BLAKE                     7698
      2450 CLARK                     7782
      3000 SCOTT                     7788
      5000 KING                      7839
      1500 TURNER                    7844
      1100 ADAMS                     7876
已选择 14 行。
```

其实，例 1-3 的显示结果与例 1-2 完全相同，只是列的显示次序不同而已。

1.3 如何书写查询语句

到目前为止我们只学习了两个关键字（Keyword）SELECT 和 FROM。Oracle 规定组成 SQL 语句的关键字是不能缩写的，也就是您不能在查询语句中将 SELECT 写成 SEL 或 SELEC 等，不能把 FROM 写成 FRO 或 FR 等，也不能把一个关键字分开书写，即关键字必须原样书写。

可以用大写、小写或大小写混写来书写 SQL 语句，如例 1-4 所示。

例 1-4

```
SQL> select Sal, EName, EMPNO
  2  From Emp;
```

例 1-4 结果

```
       SAL ENAME                    EMPNO
---------- -------------------- ----------
       800 SMITH                     7369
      1600 ALLEN                     7499
      1250 WARD                      7521
      2975 JONES                     7566
      1250 MARTIN                    7654
      2850 BLAKE                     7698
      2450 CLARK                     7782
      3000 SCOTT                     7788
      5000 KING                      7839
      1500 TURNER                    7844
      1100 ADAMS                     7876
已选择 14 行。
```

可以看出，例 1-4 显示的结果与例 1-3 完全相同。

为了增加可读性，本书所有的关键字尽量采用大写英文字母，其他的部分用小写，这也是 Oracle 公司推荐的。

既可以像例 1-3 那样把一个 SQL 语句放在多行上，也可以把一个 SQL 语句写在一行上，如例 1-5 所示。

例 1-5

```
SQL> select sal, ename, empno from emp;
```

例 1-5 结果

```
       SAL ENAME                    EMPNO
---------- -------------------- ------
       800 SMITH                  7369
      1600 ALLEN                  7499
      1250 WARD                   7521
      2975 JONES                  7566
      1250 MARTIN                 7654
      2850 BLAKE                  7698
      2450 CLARK                  7782
      3000 SCOTT                  7788
      5000 KING                   7839
      1500 TURNER                 7844
      1100 ADAMS                  7876
已选择 14 行。
```

可以看出，例 1-5 显示的结果也与例 1-3 完全相同，只是易读性不如例 1-3。

因此，为了增加可读性，应该把 SQL 语句中的每一个子句写在一行上，且最好以缩进法来书写例 1-6 的 SQL 语句。

注意：

一个完整的 SQL 命令叫语句（statement），每一个关键字和后面跟着的选项叫子句（clause），例如，“SELECT * FROM emp;”叫语句，而“SELECT *”叫子句，“FROM emp”也叫子句。

例 1-6

```
SQL> SELECT empno, ename, sal,
  2         deptno, job
  3  FROM emp;
```

例 1-6 结果

```
     EMPNO ENAME                       SAL     DEPTNO JOB
---------- -------------------- ---------- ---------- ---------
      7369 SMITH                       800         20 CLERK
      7499 ALLEN                      1600         30 SALESMAN
      7521 WARD                       1250         30 SALESMAN
      7566 JONES                      2975         20 MANAGER
      7654 MARTIN                     1250         30 SALESMAN
```

```
      7698 BLAKE                    2850          30 MANAGER
      7782 CLARK                    2450          10 MANAGER
      7788 SCOTT                    3000          20 ANALYST
      7839 KING                     5000          10 PRESIDENT
      7844 TURNER                   1500          30 SALESMAN
      7876 ADAMS                    1100          20 CLERK
已选择 14 行。
```

从例 1-6 的语句中可以很容易地看出，第 1 行和第 2 行为 SELECT 子句，第 3 行为 FROM 子句。例 1-6 的查询结果显示了 emp 表中每个员工的工号（empno）、名字（ename）、工资（sal）、部门号（deptno）和职位（job）。

1.4 列标题和数据的默认显示格式

在本章开始时，已经看到了由于字符集的不同使日期型的显示有些问题，为了解决这个问题可以使用例 1-7 的 SQL 语句。

例 1-7

```
SQL> alter session
  2  set NLS_DATE_LANGUAGE = 'AMERICAN';
```

例 1-7 结果

```
会话已更改。
```

为了使显示更加清晰，可以使用例 1-8 和例 1-9 的 SQL*Plus 格式化语句。

例 1-8

```
SQL> col hiredate for a15
```

例 1-9

```
SQL> COL ENAME FOR A8
```

注意：

如果对以上的 SQL 和 SQL*Plus 命令不太理解，请不要担心，以后还要详细介绍。

SQL*Plus 默认的列标题结果显示是：

- 字符和日期型数据为左对齐。
- 而数字型数据为右对齐。

可以使用例 1-10 的例子来验证以上的结论。

例 1-10

```
SQL> SELECT empno, ename, sal,
  2         hiredate, job
  3  FROM emp;
```

例 1-10 结果

```
     EMPNO  ENAME                SAL  HIREDATE           JOB
----------  --------  ----------  ---------------  ---------
      7369  SMITH                800  17-DEC-80          CLERK
      7499  ALLEN               1600  20-FEB-81          SALESMAN
      7521  WARD                1250  22-FEB-81          SALESMAN
      7566  JONES               2975  02-APR-81          MANAGER
      7654  MARTIN              1250  28-SEP-81          SALESMAN
      7698  BLAKE               2850  01-MAY-81          MANAGER
      7782  CLARK               2450  09-JUN-81          MANAGER
      7788  SCOTT               3000  19-APR-87          ANALYST
      7839  KING                5000  17-NOV-81          PRESIDENT
      7844  TURNER              1500  08-SEP-81          SALESMAN
      7876  ADAMS               1100  23-MAY-87          CLERK
已选择 14 行。
```

由于在运行例 1-10 的查询语句之前使用了例 1-7~例 1-9 的命令，所以例 1-10 显示输出的 HIREDATE 一列已经很整齐了。

1.5　如何在SQL语句中使用算术表达式

可以在 SQL 语句中使用表达式。在表达式中可以使用“+”、“-”、“*”、“/”4 种运算符，它们分别代表加、减、乘、除。

设想一下，在某一天有社会团体和工会团体参观你们公司，您的老板为了树立公司的光辉形象，他要求您把午餐和茶水的费用加到员工的工资里并打印一张工资清单。于是您大概算了一下，其费用约合 500 元/月。之后您用例 1-11 的查询得到了老板所要的员工的工资清单。

例 1-11

```
SQL> SELECT empno, ename, sal, 500+sal
  2  FROM emp;
```

例 1-11 结果

```
     EMPNO  ENAME             SAL     500+SAL
----------  --------  ----------  ----------
      7369  SMITH             800        1300
      7499  ALLEN            1600        2100
      7521  WARD             1250        1750
      7566  JONES            2975        3475
      7654  MARTIN           1250        1750
      7698  BLAKE            2850        3350
```

```
      7782 CLARK          2450       2950
      7788 SCOTT          3000       3500
      7839 KING           5000       5500
      7844 TURNER         1500       2000
      7876 ADAMS          1100       1600
已选择 14 行。
```

当您把例 1-11 显示的结果拿给老板时，老板认为应该用年薪并且不能列出原工资。因此，可以将例 1-11 的查询语句做一些修改并产生例 1-12 的 SQL 语句。

例 1-12

```
SQL> SELECT empno, ename, 500+sal*12
  2  FROM emp;
```

例 1-12 结果

```
     EMPNO ENAME      500+SAL*12
---------- --------- -----------
      7369 SMITH           10100
      7499 ALLEN           19700
      7521 WARD            15500
      7566 JONES           36200
      7654 MARTIN          15500
      7698 BLAKE           34700
      7782 CLARK           29900
      7788 SCOTT           36500
      7839 KING            60500
      7844 TURNER          18500
      7876 ADAMS           13700
已选择 14 行。
```

看到例 1-12 显示的结果让您大吃一惊，很显然这并不是您的老板所要的结果。其原因是运算符的优先级在作怪。

算术运算符的优先级是：

- 先乘除后加减。
- 在表达式中同一优先级的运算符计算次序是从左到右。
- 如果使用了括号，括号中的运算优先。
- 如果有多重括号嵌套，内层括号中的运算优先。

因此您不得不再次重新书写例 1-13 的 SQL 语句。

例 1-13

```
SQL> SELECT empno, ename, (500+sal)*12
  2  FROM emp;
```

例 1-13 结果

```
     EMPNO ENAME      (500+SAL)*12
---------- --------  ------------
      7369 SMITH            15600
      7499 ALLEN            25200
      7521 WARD             21000
      7566 JONES            41700
      7654 MARTIN           21000
      7698 BLAKE            40200
      7782 CLARK            35400
      7788 SCOTT            42000
      7839 KING             66000
      7844 TURNER           24000
      7876 ADAMS            19200
已选择 14 行。
```

毫无疑问，例 1-13 显示的结果就是老板所要的员工的工资清单。虽然您得到了正确的结果，但显示的列标题确实是令人费解，特别是对您的老板和那些团体的头头们，因为他们对 Oracle 数据库可能是一窍不通。

1.6　如何在SQL语句中使用列的别名

为了让老板明白，您不辞辛苦地再次修改了例 1-13 中的查询语句，这次您使用了列的别名，写出了例 1-14 的 SQL 语句。

例 1-14

```
SQL> SELECT empno AS "Employee Number",
  2         ename name, (500+sal)*12  "Annual Salary"
  3  FROM emp;
```

例 1-14 结果

```
Employee Number NAME                    Annual Salary
--------------- -------------------- -------------
           7369 SMITH                        15600
           7499 ALLEN                        25200
           7521 WARD                         21000
           7566 JONES                        41700
           7654 MARTIN                       21000
           7698 BLAKE                        40200
           7782 CLARK                        35400
           7788 SCOTT                        42000
```

```
            7839 KING                         66000
            7844 TURNER                       24000
            7876 ADAMS                        19200
...
已选择 14 行。
```

上面例 1-14 显示的结果正是您的老板所要的。

Oracle 在为列命名时大量使用缩写方式，这些以缩写方式表示列名的好处之一是它减少了输入量，但对非计算机专业的人员来说，这些缩写方式的列名如同天书一般。有没有一种方法既能满足 Oracle 专业人员习惯于使用缩写的癖好，又使非计算机专业人员看到显示的结果时一目了然呢？为列起一个别名就解决了这一难题（就像在上面的例子中所做的那样）。

给一列起一个别名的方法很简单。您只需在列名和别名之间放上 AS 或空格就可以了。虽然从语句的易读性角度来说应该使用 AS 关键字，但是使用这一关键字需要多输入两个字符，所以您会发现许多的 Oracle 专业人员很少使用 AS 这一关键字。

别名对处理表达式表示的列非常有用。您可能已经注意到了，当别名没有被双引号括起来时，其显示结果为大写。如果别名中包含了特殊字符，或想让别名原样显示，您就要使用双引号把别名括起来。

1.7 连接运算符

您的老板对您为他做的员工工资清单很满意，但他想使显示的结果是一个完整的英文句子，而且表达式的列标为“Employee’s Salary”。这样，那些团体的头头们更容易理解。为此，您不得不再一次修改查询语句，写下了例 1-15 的 SQL 语句。

例 1-15

```
SQL> SELECT ename ||' annual salary is '||(500+sal)*12 "Employee's Salary"
  2  FROM emp;
```

例 1-15 结果

```
Employee's Salary
-----------------------------
SMITH annual salary is 15600
ALLEN annual salary is 25200
WARD annual salary is 21000
JONES annual salary is 41700
MARTIN annual salary is 21000
BLAKE annual salary is 40200
CLARK annual salary is 35400
SCOTT annual salary is 42000
KING annual salary is 66000
```

```
TURNER annual salary is 24000
ADAMS annual salary is 19200
…
```

当您把这份报告呈给老板时，老板灿烂的微笑已经告诉您，这正是他所期望的报告。如果您使用的是中文系统，也可以使用例 1-16 的带有中文的查询语句。

例 1-16

```
SQL> SELECT ename ||' 年薪为：'||(500+sal)*12 "员工的年薪"
  2  FROM emp;
```

例 1-16 结果

```
员工的年薪
--------------------
SMITH 年薪为：15600
ALLEN 年薪为：25200
WARD 年薪为：21000
JONES 年薪为：41700
MARTIN 年薪为：21000
BLAKE 年薪为：40200
CLARK 年薪为：35400
SCOTT 年薪为：42000
KING 年薪为：66000
TURNER 年薪为：24000
ADAMS 年薪为：19200
JAMES 年薪为：17400
FORD 年薪为：42000
MILLER 年薪为：21600
已选择 14 行。
```

在这一查询语句中使用了文本字符串（literal）和连接运算符。

文本字符串是包含在 SELECT 子句中的字符、数字或表达式，而不是任何的列名或列的别名。如果文本字符是日期型和字符型，就必须将它们用单引号括起来。每个字符串在每行输出结果中都输出一次。

连接运算符由两个竖线（||）表示，它用于把一个或多个列或字符串连接在一起。

1.8　DISTINCT运算符

设想您刚刚被公司聘为 Oracle 数据库管理员（DBA），想知道您的公司究竟有多少个部门，可能发出例 1-17 的查询语句。

例 1-17

```
SQL> select deptno
  2  from emp;
```

例 1-17 结果

```
    DEPTNO
-----------
        20
        30
        30
        20
        30
        30
        10
        20
        10
        30
…
已选择 14 行。
```

Oracle 显示了全部的记录，这是 Oracle 默认的显示方式。如果您所在的公司是一个大型跨国企业，它雇佣了一百多万名员工，可以想象这一查询会造成什么样的后果。

可以用 DISTINCT 帮助您去掉重复的行。请看例 1-18 的查询。

例 1-18

```
SQL> SELECT DISTINCT deptno
  2  FROM emp;
```

例 1-18 结果

```
   DEPTNO
----------
       10
       20
       30
```

很显然，例 1-18 显示的结果要比例 1-17 的结果清楚多了。

注意：

当查询比较大的表时应尽可能地避免使用 DISTINCT，因为 Oracle 系统是通过排序的方式来完成 DISTINCT 这一功能的，所以它会造成 Oracle 系统的效率降低。通常您可以使用不同的查询语句来完成同样的工作，如可以使用例 1-19 的查询语句来得到所需的部门的信息。

例 1-19

```
SQL> SELECT deptno
  2  FROM dept;
```

例 1-19 结果

```
   DEPTNO
----------
       10
       20
       30
       40
```

例 1-19 的查询与前面的查询一样得到了所有的部门号，但对 Oracle 系统没什么冲击，因为部门号（deptno）在 dept 表中本身就是唯一的。

DISTINCT 可以作用于多列，此时显示的结果为每一种列组合只显示一行，如例 1-20。

例 1-20

```
SQL> SELECT DISTINCT deptno, job
  2  FROM emp;
```

例 1-20 结果

```
   DEPTNO JOB
---------- ---------
       10 CLERK
       10 MANAGER
       10 PRESIDENT
       20 ANALYST
       20 CLERK
       20 MANAGER
       30 CLERK
       30 MANAGER
       30 SALESMAN
已选择 9 行。
```

到此为止，我们已经讨论了最基本的查询语句。下面将给出基本查询语句的格式。

1.9　基本查询语句的格式

在这一章中我们只学习了一个基本的查询语句，它只包含了两个子句 SELECT 和 FROM，其格式如下：

SELECT　*|{[DISTINCT] 列表,…}

FROM　表名;

其中，列表的格式为：

列名|表达式 [别名]

“SELECT * FROM 表名;”为从表名所指定的表中选择所有的列；“SELECT DISTINCT 列表,… FROM 表名;”为从表名所指定的表中选择列表所规定的列，但不显示重复的数据行（记录）。

1.10 应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 在查询语句中哪两个子句是必需的？
- 什么是投影操作？
- 如何在查询语句中选择特定的列及列的顺序？
- 书写查询语句的约定。
- 什么是语句？
- 什么是子句？
- 列标题的默认显示格式。
- 字符和日期型数据的默认显示格式。
- 数字型数据的默认显示格式。
- SQL 语句中的算术表达式及它们的优先级。
- SQL 语句中列的别名及其使用。
- 如何使用文本字符串（literal）和连接运算符（||）？
- 如何使用 DISTINCT 关键字？使用它可能产生的问题有哪些？

第2章

限制性查询和数据的排序

在第 1 章中所有的查询结果都是显示整个表中所有的记录。如果所操作的表很大，如表中包含了一百多万行的数据，从这样的查询结果中很难找到所需要的信息。这也可能就是引入限制性查询和排序的原因。

2.1　如何限制所选择的数据行

设想一下，公司因受互联网泡沫破裂和 9.11 恐怖袭击的双重打击，已经连续几个季度亏损，老板不得不进行优化组合（解雇一批员工），以节省开销。老板当然想从高收入者开刀，他让您打印一份工资在 1500 元（包括 1500 元）或高于 1500 元的员工的清单，以决定哪些“无用之人”应该炒鱿鱼。您可以使用例 2-1 的查询语句来完成老板的重托。

例 2-1

```
SQL> SELECT empno, ename, sal
  2  FROM emp
  3  WHERE sal >= 1500;
```

例 2-1 结果

```
     EMPNO ENAME                      SAL
---------- -------------------- ---------
      7499 ALLEN                     1600
      7566 JONES                     2975
      7698 BLAKE                     2850
      7782 CLARK                     2450
      7788 SCOTT                     3000
      7839 KING                      5000
      7844 TURNER                    1500
      7902 FORD                      3000
已选择 8 行。
```

例 2-1 显示的结果给出了工资为 1500 元或以上的所有员工的工号（empno）、名字（ename）和工资（sal）。这张清单上列出了所有可能被解雇的候选员工。

在例 2-1 的查询语句中，使用了 WHERE 子句来限制哪些行（记录）要显示，这在关

系数据库中称为选择（SELECTION）操作。WHERE 子句跟在 FROM 子句之后。

- WHERE 是关键字，其后跟限制条件。
- 条件是由列名、字符串、算术表达式、常量和比较运算符组成的。

2.2 比较运算符

可以使用比较运算符（operators）来构造条件。

条件的格式为：

表达式　operator　表达式

Oracle 提供了>（大于）、>=（大于等于）、<（小于）、<=（小于等于）、=（等于）、<>或！=（不等于）6 个常用的比较运算符。

除了以上的比较运算符外，Oracle 还提供了 BETWEEN AND、IN 和 LIKE 3 个比较运算符。

以下几节将通过一些例子来解释这些比较运算符的用法。

2.3 如何使用BETWEEN AND比较运算符

假设您的公司规定工资在 2900 元或以上的员工的聘用和解雇要由董事会来决定。因此，老板不希望您给他的清单中包含工资在 2900 元或以上的员工，于是，您写出例 2-2 的查询语句。

例 2-2

```
SQL> SELECT empno, ename, sal
  2  FROM emp
  3  WHERE sal BETWEEN 1500 AND 2900;
```

例 2-2 结果

```
     EMPNO ENAME                       SAL
---------- -------------------- ------
      7499 ALLEN                      1600
      7698 BLAKE                      2850
      7782 CLARK                      2450
      7844 TURNER                     1500
```

这样，例 2-2 显示的结果中就只包括了老板所希望见到的所有员工的清单。

BETWEEN 运算符用于测试某些值是否在指定的数值范围之内。在 BETWEEN 和 AND 之间的值称为下限，AND 之后的值称为上限。显示的结果包含下限和上限的值。该运算符不但可用于数字型数据，而且还可用于字符型和日期型数据，但这两种类型的数据必须用单引号括起来。

例如，老板想知道谁是在 1981 年 1 月 1 日和 1982 年 5 月 31 日之间加入该公司的，可

以使用例 2-3 的查询语句来得到他所需要的信息。

例 2-3

```
SQL> SELECT empno, ename, sal, hiredate
  2  FROM emp
  3  WHERE hiredate BETWEEN '01-JAN-81' AND '31-MAY-82';
```

例 2-3 结果

```
     EMPNO ENAME                    SAL HIREDATE
---------- ---------------- ---------- -------
      7499 ALLEN                   1600 20-FEB-81
      7521 WARD                    1250 22-FEB-81
      7566 JONES                   2975 02-APR-81
      7654 MARTIN                  1250 28-SEP-81
      7698 BLAKE                   2850 01-MAY-81
      7782 CLARK                   2450 09-JUN-81
      7839 KING                    5000 17-NOV-81
      7844 TURNER                  1500 08-SEP-81
      7900 JAMES                    950 03-DEC-81
      7902 FORD                    3000 03-DEC-81
      7934 MILLER                  1300 23-JAN-82
已选择 11 行。
```

该查询语句的结果显示了从 1981 年 1 月到 1982 年 5 月所聘用的员工清单。如果 HIREDATE 一列显示的为乱码，您可能需要使用例 2-4 的命令。

例 2-4

```
SQL> alter session set NLS_DATE_LANGUAGE = AMERICAN;
```

例 2-4 结果

```
会话已更改。
```

您还记得上面的 SQL 语句的用途吗？如果不记得了，请复习一下第 1 章有关该 SQL 语句的说明。

可以在 BETWEEN 之前加上 NOT，用于测试某些值是否在 BETWEEN 和 AND 指定的数值范围之内。当看了例 2-3 的结果后，您的老板突然想到，如果有一张不是在 1981 年 1 月 1 日到 1982 年 5 月 31 日加入该公司的员工的清单，就可帮助他更容易地作决策。因为这段时间公司刚刚起步，在这段时间加入公司的员工多数持有公司的股份。于是他又把这一想法告诉您。您使用了例 2-5 的查询语句以得到老板所需的信息。

例 2-5

```
SQL> SELECT empno, ename, sal, hiredate
  2  FROM emp
  3  WHERE hiredate NOT BETWEEN '01-JAN-81' AND '31-MAY-82';
```

例 2-5 结果

```
    EMPNO  ENAME                               SAL  HIRDATE
---------- ----------------- ---------- ---------
      7369  SMITH                               800  17-DEC-80
      7788  SCOTT                              3000  19-APR-87
      7876  ADAMS                              1100  23-MAY-87
```

这一 SQL 语句得到的结果为一张除了从 1981 年 1 月到 1982 年 5 月所聘用的员工以外的所有员工的清单。对这张清单上的员工作决策，您的老板就没有什么顾忌了。

2.4　在SQL语句中使用字符串和日期

您的老板考虑到目前的销售不景气还要持续一段时期，因此不需要那么多的推销员（SALESMAN）。他让您现在打印一份所有推销员的清单，于是您写下了例 2-6 的查询语句。

例 2-6

```
SQL> SELECT empno, ename, job
   2  FROM emp;
   3  WHERE JOB = 'SALESMAN';
```

例 2-6 结果

```
未选定行。
```

可是没有显示出任何结果。您一定对此感到很惊讶，因为在员工（emp）表中确实存有推销员的数据，不显示任何结果是为什么呢？

这是因为 WHERE 子句中的字符串是区分大小写的。在 WHERE 子句中字符和日期型数据要用单引号括起来，但数字型不用。日期型数据默认的格式，在 Oracle 9i 之前的版本中为 DD-MON-YY，在 Oracle 9i 中为 DD-MON-RR。

当您看完以上的解释后，就已经知道了问题所在，重新输入例 2-7 的查询语句。

例 2-7

```
SQL> SELECT empno, ename, job, sal
   2  FROM emp
   3  WHERE job ='SALESMAN';
```

例 2-7 结果

```
    EMPNO  ENAME                   JOB                       SAL
--------- ------------------ ---------------- ---------
      7499  ALLEN                  SALESMAN                 1600
      7521  WARD                   SALESMAN                 1250
      7654  MARTIN                 SALESMAN                 1250
      7844  TURNER                 SALESMAN                 1500
```

现在您终于得到了老板所要的结果。

2.5　使用IN比较运算符

如果您的老板不但不需要那么多的推销员（SALESMAN），而且也不需要那么多的文员和经理。又该如何打印这份员工的清单呢？这时可以使用 IN 操作符来帮助完成这一工作。您可以使用例 2-8 的查询语句。

例 2-8

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE job IN ('SALESMAN', 'CLERK', 'MANAGER');
```

例 2-8 结果

```
     EMPNO ENAME                      SAL JOB
---------- -------------------- --------- ------
      7369 SMITH                      800 CLERK
      7499 ALLEN                     1600 SALESMAN
      7521 WARD                      1250 SALESMAN
      7566 JONES                     2975 MANAGER
      7654 MARTIN                    1250 SALESMAN
      7698 BLAKE                     2850 MANAGER
      7782 CLARK                     2450 MANAGER
      7844 TURNER                    1500 SALESMAN
      7876 ADAMS                     1100 CLERK
      7900 JAMES                      950 CLERK
      7934 MILLER                    1300 CLERK
已选择 11 行。
```

IN 也是 SQL 中一个很有用的比较运算符。IN 用来测试某些值是否在列表中出现。在上面的 SQL 语句中，只要某一记录的 JOB 列的值等于 IN 列表中（即括号中）的任何一个，该记录就会显示出来。

另外，还可以在 IN 之前加上否定词 NOT。NOT IN 用来测试某些值是否不在列表中出现。现在进一步假设您所在的公司一直没能扭转亏损的势头，老板不得不考虑解雇更多的员工以进一步节省开销。他认为公司中除了分析员（ANALYST）以外都可以是要解雇的候选人。他让您按此要求再为他准备一张员工的清单。于是您写下了例 2-9 的查询语句。

例 2-9

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE job NOT IN ('ANALYST', 'PRESIDENT');
```

例 2-9 结果

```
     EMPNO ENAME                                   SAL JOB
---------- ------------------ ---------- ---------
      7369 SMITH                                   800 CLERK
      7499 ALLEN                                  1600 SALESMAN
      7521 WARD                                   1250 SALESMAN
      7566 JONES                                  2975 MANAGER
      7654 MARTIN                                 1250 SALESMAN
      7698 BLAKE                                  2850 MANAGER
      7782 CLARK                                  2450 MANAGER
      7844 TURNER                                 1500 SALESMAN
      7876 ADAMS                                  1100 CLERK
      7900 JAMES                                   950 CLERK
      7934 MILLER                                 1300 CLERK
已选择 11 行。
```

这个例子与前面的例子显示的结果完全相同，但使用了不同的条件。"WHERE job NOT IN（'ANALYST', 'PRESIDENT'）"告诉 Oracle 只显示那些 JOB（职位）既不是 ANALYST（分析员）也不是 PRESIDENT（老总）的记录。因为在 emp 表中只有 5 种职位，除了 ANALYST 和 PRESIDENT 就是 SALESMAN、CLERK 和 MANAGER 了。这一例子也告诉我们可以由不同的查询语句来得到相同的结果。

如果仔细回忆一下老板的观点，即他认为公司中除了分析员（ANALYST）以外都可以是要解雇的候选人，就会发现例 2-9 的查询语句好像与他的要求有点出入，因为他并没有说要解雇的候选人中不包含总裁（PRESIDENT），即老板自己。

2.6 使用LIKE比较运算符

我们知道人的记忆是非常有趣的，想记住的事很难记得住，但想忘记的事却永远也忘不掉。假设老板让您打印一份所有推销员的清单时，您不记得 SALESMAN 的准确拼写，但还记得它的前 3 个字符 SAL，此时该怎么办呢？可以在您的查询语句中使用 LIKE 运算符。例如，可以使用例 2-10 的 SQL 语句。

例 2-10

```
SQL> SELECT  empno, ename, sal, job
   2  FROM emp
   3  WHERE job LIKE 'SAL%';
```

例 2-10 结果

```
     EMPNO ENAME                                 SAL JOB
---------- -------------------- ---------- ----
      7499 ALLEN                                1600 SALESMAN
```

```
     7521 WARD                        1250 SALESMAN
     7654 MARTIN                      1250 SALESMAN
     7844 TURNER                      1500 SALESMAN
```

您可以用 LIKE 运算符进行通配符查询。通配符的英文原文为 wildcard，该词的原意为扑克牌中的 2 或王，因为它们可以代替任何其他的牌，所以称为 wildcard。

LIKE 运算符可以使用以下两个通配符“%”和“-”。其中：

- “%”代表 0 个或多个字符。
- “-”代表一个且只能是一个字符。

如果只记得 SALESMAN 的第 1 个字符为 S，第 3 个字符为 L，第 5 个字符为 S，那该如何处理呢？您可使用例 2-11 的查询语句。

例 2-11

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE job LIKE 'S_L_S%';
```

例 2-11 结果

```
    EMPNO ENAME                      SAL JOB
---------- -------------------- ---------- ---------
     7499 ALLEN                       1600 SALESMAN
     7521 WARD                        1250 SALESMAN
     7654 MARTIN                      1250 SALESMAN
     7844 TURNER                      1500 SALESMAN
```

从例 2-11 的查询语句可以看出，通过在 LIKE 表达式中使用不同的通配符“%”和“-”的组合，可以构造出相当复杂的限制条件。

另外，LIKE 运算符还可以帮助您简化某些 WHERE 子句。例如，要显示在 1981 年雇用的所有员工的清单，即可以使用例 2-12 的查询语句。

例 2-12

```
SQL> SELECT empno, ename, sal, hiredate
  2  FROM emp
  3  WHERE hiredate LIKE '%81';
```

例 2-12 结果

```
    EMPNO ENAME                    SAL HIREDATE
---------- ------------------ ---------- ---------
     7499 ALLEN                     1600 20-FEB-81
     7521 WARD                      1250 22-FEB-81
     7566 JONES                     2975 02-APR-81
     7654 MARTIN                    1250 28-SEP-81
     7698 BLAKE                     2850 01-MAY-81
     7782 CLARK                     2450 09-JUN-81
```

```
      7839 KING                             5000 17-NOV-81
      7844 TURNER                           1500 08-SEP-81
      7900 JAMES                            950  03-DEC-81
      7902 FORD                             3000 03-DEC-81
已选择 11 行。
```

如果要查询的字符串中含有“-”或“%”，又该如何处理呢？

2.7 如何使用转义操作符

要查询的字符串中含有“-”或“%”时，可以使用转义（escape）关键字实现查询。为了进行练习，必须先创建一个临时的表，之后再往该表中插入 1 行记录，其值包含通配符。现在您可能不十分理解例 2-13 和例 2-14 的 SQL 语句。没有问题，您只要按照例子输入就可以了。

例 2-13

```
SQL> CREATE TABLE dept_temp
   2  AS
   3  SELECT *
   4  FROM dept;
```

例 2-13 结果

```
表已创建。
```

例 2-14

```
SQL> INSERT INTO dept_temp
   2  VALUES (88,'IT_RESEARCH','BEIJING');
```

例 2-14 结果

```
已创建 1 行。
```

现在就可以输入例 2-15 的查询语句来显示其部门名（dname）以 IT_开始的所有数据行。

例 2-15

```
SQL> SELECT *
   2  FROM dept_temp
   3  WHERE dname LIKE 'IT\_%' escape '\';
```

例 2-15 结果

```
   DEPTNO DNAME                          LOC
---------- ------------------------ -------
       88 IT_RESEARCH                    BEIJING
```

在例 2-15 的查询语句中，您定义“\”为转义符，即在“\”之后的“_”字符已不是通配符了，而是它本来的含义，即下划线。因此该查询的结果为：前两个字符为“IT”，第

3 个字符为“_”，后跟任意字符。

没有必要一定使用“\”字符作为转义符，完全可以使用任何您感兴趣的字符作为转义符。许多 Oracle 的专业人员之所以经常使用“\”字符作为转义符，是因为该字符在 UNIX 操作系统和 C 语言中就是转义符。

为了验证以上的论述，您可以输入例 2-16 的查询语句。

例 2-16

```
SQL> SELECT *
  2  FROM dept_temp
  3  WHERE dname LIKE 'IT~_%' escape '~';
```

例 2-16 结果

```
    DEPTNO DNAME          LOC
---------- -------------- -------
        88 IT_RESEARCH    BEIJING
```

在例 2-16 的查询语句中，将“~”定义为转义字符，但是例 2-16 的显示结果与例 2-15 的完全相同。

建议：

最好不要将在 SQL 和 SQL*Plus 中有特殊含义的字符定义为转义符，否则使您的 SQL 语句变得很难理解。

2.8　ORDER BY 子句

还记得在本章开始时的例子吗？如果您忘记了，请您翻回到本章开始的地方，重新阅读一下。现在您的老板要您重新打印那份工资清单，但是要按工资数额由大到小排序。您翻一翻 SQL 的手册，发现 ORDER BY 子句是用来排序的。于是您试着写下了例 2-17 的查询语句。

例 2-17

```
SQL> SELECT empno, ename, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4  ORDER BY sal;
```

例 2-17 结果

```
     EMPNO ENAME                       SAL
---------- -------------------- ------
      7844 TURNER                     1500
      7499 ALLEN                      1600
```

```
      7782 CLARK                    2450
      7698 BLAKE                    2850
      7566 JONES                    2975
      7788 SCOTT                    3000
      7902 FORD                     3000
      7839 KING                     5000
已选择 8 行。
```

其显示的结果虽然是排序后的，但是顺序是由小到大。设想一下您所在的公司可能是一个大型的企业，可能有几十万名员工，在这种情况下这个清单可能为几百或几千页，因此老板更希望先看到高薪员工。于是您又一次地重写了例 2-18 的查询语句。

例 2-18

```
SQL> SELECT empno, ename, sal
   2  FROM emp
   3  WHERE sal >= 1500
   4  ORDER BY sal DESC;
```

例 2-18 结果

```
     EMPNO ENAME                     SAL
---------- -------------------- ------
      7839 KING                     5000
      7788 SCOTT                    3000
      7902 FORD                     3000
      7566 JONES                    2975
      7698 BLAKE                    2850
      7782 CLARK                    2450
      7499 ALLEN                    1600
      7844 TURNER                   1500
已选择 8 行。
```

例 2-18 的结果正是老板所需要的清单。在这个查询语句中使用了 DESC 关键字。

利用 ORDER BY 子句对查询的结果进行排序。ASC（ascending order）为升序排序（默认），因此，ASC 在实际的 SQL 语句中很少见到。您可以对数字型、日期型和字符型数据进行排序。默认时，数字型和日期型数据的顺序为从小到大。字符型数据的顺序是按 ASCII 码的次序，即从 A 到 Z。DESC（descending order）为降序排序。

如果在查询语句中不使用 ORDER BY 子句，则查询结果的次序是不确定的，即您写了两个完全相同的查询语句，其执行结果的次序可能是不一样的。

需要注意的是，如果使用了 ORDER BY 子句，则该子句一定是 SQL 语句的最后一个子句。

2.9　在ORDER BY子句中使用别名或表达式

还记得在第 1 章中的例 1-11~例 1-14 吗？现在我们将第 1 章的例 1-14 的 SQL 语句重写，如例 2-19 所示。

例 2-19

```
SQL>SELECT empno AS "Employee Number",ename name,(500+sal)*12 "Annual Salary"
 2 FROM emp
```

该 SQL 语句显示的结果没什么规律，而老板更喜欢把高薪者排在前面。由于年薪一列是表达式，应该用别名来排序。您可以使用例 2-20 的 SQL 语句来满足老板的要求。

例 2-20

```
SQL>SELECT empno AS "Employee Number", ename name, (500+sal)*12 "Annual Salary"
  2   FROM emp
  3   ORDER BY "Annual Salary" DESC;
```

例 2-20 结果

```
Employee Number NAME       Annual Salary
--------------- ---------- -------------
           7839 KING               66000
           7788 SCOTT              42000
           7902 FORD               42000
           7566 JONES              41700
           7698 BLAKE              40200
           7782 CLARK              35400
           7499 ALLEN              25200
           7844 TURNER             24000
           7934 MILLER             21600
           7521 WARD               21000
           7654 MARTIN             21000
           7876 ADAMS              19200
           7900 JAMES              17400
           7369 SMITH              15600
已选择 14 行。
```

除了如例 2-20 所示在 ORDER BY 子句后面使用别名外，还可以在 ORDER BY 子句后面跟表达式。您可以输入例 2-21 的 SQL 语句。

例 2-21

```
SQL>SELECT empno AS "Employee Number", ename name, (500+sal)*12 "Annual Salary"
  2  FROM emp
  3  ORDER BY  (500+sal)*12 DESC;
```

例 2-21 结果

```
Employee Number NAME       Annual Salary
--------------- ---------- -------------
           7839 KING               66000
           7788 SCOTT              42000
           7902 FORD               42000
           7566 JONES              41700
           7698 BLAKE              40200
           7782 CLARK              35400
           7499 ALLEN              25200
           7844 TURNER             24000
           7934 MILLER             21600
           7521 WARD               21000
           7654 MARTIN             21000
           7876 ADAMS              19200
           7900 JAMES              17400
           7369 SMITH              15600
已选择 14 行。
```

您得到了与例 2-20 完全相同的结果。只是例 2-21 查询语句中的 ORDER BY 子句看起来不如上一个例子容易理解。

2.10　在ORDER BY子句中使用列号

此外我们也可以用例 2-22 的查询语句来完成相同的工作，只是看上去更加令人费解。这里 3 表示第 3 列，所以 ORDER BY 3 就是按第 3 列排序。

例 2-22

```
SQL> SELECT empno "Employee Number",ename name,(500+sal)*12 "Annual Salary"
  2  FROM emp
  3  ORDER BY 3 DESC;
```

例 2-22 结果

```
Employee Number NAME       Annual Salary
--------------- ---------- -------------
           7839 KING               66000
           7788 SCOTT              42000
           7902 FORD               42000
           7566 JONES              41700
           7698 BLAKE              40200
           7782 CLARK              35400
```

```
            7499 ALLEN                25200
            7844 TURNER               24000
            7934 MILLER               21600
            7521 WARD                 21000
            7654 MARTIN               21000
            7876 ADAMS                19200
            7900 JAMES                17400
            7369 SMITH                15600
已选择 14 行。
```

应该在 SQL 语句中尽可能地不使用 ORDER BY 子句的这种用法，因为这种用法的易读性实在太差了。在不少有关 Oracle SQL 的书中根本就没有介绍 ORDER BY 子句的这一用法。尽管如此，由于该用法可以减少输入，特别是当放在 ORDER BY 子句之后的列名或表达式很长时，所以还是有人使用这种用法。本书介绍这一用法的目的是，当看到 SQL 语句中包含了这一用法时，您可以理解它，但并不鼓励使用。

2.11　在ORDER BY子句中使用多列

您也可以对多列进行排序，这些列的排序既可以按升序也可以按降序。假设想以这样的方式来显示员工的名字、职位和工资，则应首先按职位由 A 到 Z 排序，之后再按工资由高到低排序。于是可以使用例 2-23 的查询语句来达到这一目的。

例 2-23

```
SQL> SELECT ename, job, sal
  2  FROM emp
  3  ORDER BY job, sal DESC;
```

例 2-23 结果

```
ENAME      JOB              SAL
---------- --------- ----------
SCOTT      ANALYST         3000
FORD       ANALYST         3000
MILLER     CLERK           1300
ADAMS      CLERK           1100
JAMES      CLERK            950
SMITH      CLERK            800
JONES      MANAGER         2975
BLAKE      MANAGER         2850
CLARK      MANAGER         2450
KING       PRESIDENT       5000
ALLEN      SALESMAN        1600
TURNER     SALESMAN        1500
```

```
WARD        SALESMAN        1250
MARTIN      SALESMAN        1250
已选择 14 行。
```

2.12 在ORDER BY子句中使用在SELECT列表中没有的列

您还可以用不在 SELECT 列表中的列来排序。例如，可以输入例 2-24 的查询语句。

例 2-24

```
SQL> SELECT ename, job, sal
  2  FROM emp
  3  ORDER BY empno;
```

例 2-24 结果

```
ENAME       JOB          SAL
---------- --------- ----------
SMITH       CLERK            800
ALLEN       SALESMAN     1600
WARD        SALESMAN     1250
JONES       MANAGER      2975
MARTIN      SALESMAN         1250
BLAKE       MANAGER      2850
CLARK       MANAGER      2450
SCOTT       ANALYST      3000
KING        PRESIDENT    5000
TURNER      SALESMAN         1500
ADAMS       CLERK            1100
JAMES       CLERK            950
FORD        ANALYST      3000
MILLER      CLERK        1300
已选择 14 行。
```

但是您应该尽可能避免使用这种排序的方法。因为用这种方法得到的结果其他人很难看得懂。我相信随着时间的流逝，您自己也会看不懂的。

2.13 扩充后的查询语句的格式

本章对基本的查询语句进行了扩充，加入了 WHERE 子句和 ORDER BY 子句。它的格式如下：

```
SELECT   *|{[DISTINCT] 列表,…}
FROM   表名
[WHERE 条件]
[ORDER BY {列名|别名|表达式,…}[ASC|DESC]];
```

其中，条件由列名、文字串、算术表达式、常量和比较运算符 5 部分组成。

需要注意的是，ORDER BY 子句一定要放在 SQL 语句的最后。

2.14　应该掌握的内容

在学习下一章之前，请检查一下是否已经掌握了以下的内容：

- 什么是关系数据库中的选择（selection）操作？
- 如何利用 WHERE 子句来限制所选的数据行？
- 6 个常用的比较运算符（operators）。
- 如何构造 WHERE 子句中的条件？
- BETWEEN AND 和 IN 比较运算符（operators）。
- NOT 运算符的使用。
- 字符串和日期数据在 SQL 语句中的表示。
- LIKE 比较运算符（operators）和通配符（wildcard）。
- 转义（escape）字符的应用。
- ORDER BY 子句的简单使用。
- 使用别名或表达式进行排序。
- 使用列号进行排序。
- 使用多列来排序。
- 使用不在 SELECT 列表中的列来排序。

第3章

常用的 SQL*Plus 命令

当输入 SQL 语句时，该语句被存在 SQL 缓冲区中（一个内存区），这个 SQL 缓冲区很小，只能存一个 SQL 语句，当下一条 SQL 语句输入时，原来在缓冲区中的 SQL 语句被覆盖掉。SQL*Plus 是一个工具（环境）。正像我们所看到的，我们可以用它来输入 SQL 语句。为了有效地输入和编辑 SQL 语句，SQL*Plus 还提供了一些常用的命令。与 SQL 语句不同的是 SQL*Plus 命令是可以缩写的。下面就简单地介绍一些常用的 SQL*Plus 命令。

3.1　DESC[RIBE]命令

一般在操作表之前总是想知道表的结构，可以使用 DESC[RIBE]命令来实现。您可以使用例 3-1 的命令来显示 emp 表的结构。

例 3-1

```
SQL> DESC emp
```

例 3-1 结果

```
名称                                      是否为空? 类型
----------------------------------------- -------- --------
EMPNO                                     NOT NULL NUMBER(4)
ENAME                                              VARCHAR2(10)
JOB                                                VARCHAR2(9)
MGR                                                NUMBER(4)
HIREDATE                                           DATE
SAL                                                NUMBER(7,2)
COMM                                               NUMBER(7,2)
DEPTNO                                             NUMBER(2)
```

从例 3-1 的显示结果可知，所谓一个表的结构，就是该表中包含了多少个列，每一列的数据类型和它的最大长度，以及该列是否可以为空（NULL）（也称为约束，我们将在后面的章节中详细介绍）。

例 3-1 显示的结果告诉我们，emp 表中包含了 8 列，其中只有 empno 列不能为空。各列的数据类型如下：

- EMPNO 列为整数，最大长度为 4 位。

- ENAME 列为变长字符型，最大长度为 10 个字符。
- JOB 列也为变长字符型，最大长度为 9 个字符。
- MGR 列为整数，最大长度为 4 位。
- HIREDATE 列为日期型（将在后面的章节中详细介绍这种数据类型）。
- SAL 列为浮点数（即包含小数的数），最大长度为 7 位，其中有两位是小数。
- COMM 列也为浮点数，最大长度也为 7 位，其中有两位是小数。
- DEPTNO 列为整数，最大长度为两位。

也可以使用例 3-2 的命令来显示 dept 表的结构。

例 3-2

```
SQL> DESC dept
```

例 3-2 结果

```
名称                                      是否为空? 类型
----------------------------------------- -------- ------------
DEPTNO                                    NOT NULL NUMBER(2)
DNAME                                              VARCHAR2(14)
LOC                                                VARCHAR2(13)
```

例 3-2 显示的结果告诉我们，dept 表中包含了 3 列，其中只有 DEPTNO 这一列不能为空。各列的数据类型如下：

- DEPTNO 列为整数，最大长度为两位。
- DNAME 列为变长字符型，最大长度为 14 个字符。
- LOC 列也为变长字符型，最大长度为 13 个字符。

从上面的例 3-1 和例 3-2 可以看出，SQL*Plus 命令的结尾处可以不使用分号（;）。

DESC[RIBE]命令是经常使用的 SQL*Plus 命令。一般有经验的开发人员（程序员）在使用 SQL 语句开发程序之前，都要使用 DESC[RIBE]命令来查看 SQL 语句要操作的表的结构，因为一旦开发人员清楚了所操作的表的结构，可以明显地减少程序出错的概率。

3.2　SET LINE[SIZE]{80|n}命令

另一个有用的SQL*Plus命令是SET LINE[SIZE] {80| n}，其中n为自然数，80 为默认值。该命令是将显示屏的显示输出置为n个字符宽，80 个字符为此命令的默认显示宽度。

如果想使用例 3-3 的 SQL 语句来显示 emp 表中所有的列，会发现显示的结果很难看懂。

例 3-3

```
SQL> SELECT *
  2  FROM emp;
```

例 3-3 结果

```
     EMPNO ENAME      JOB              MGR HIREDATE          SAL       COMM
---------- ---------- --------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7369 SMITH      CLERK           7902 17-12月-80        800
        20

      7499 ALLEN      SALESMAN        7698 20-2 月 -81       1600        300
        30

      7521 WARD       SALESMAN        7698 22-2 月 -81       1250        500
        30

     EMPNO ENAME      JOB              MGR HIREDATE          SAL       COMM
---------- ---------- --------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7566 JONES      MANAGER         7839 02-4 月 -81       2975
        20

      7654 MARTIN     SALESMAN        7698 28-9 月 -81       1250       1400
        30

      7698 BLAKE      MANAGER         7839 01-5 月 -81       2850
        30 ……
```

如果您的显示屏幕足够大，就可以使用 SQL*Plus 命令 SET LINE 100，如例 3-4 所示。

例 3-4

```
SQL> SET line 100
```

此时，您就会发现其显示输出好懂得多了，因为每一行数据都显示在同一行上，而不是像例 3-3 显示的结果那样，同一行的数据显示在两个不同行上。

3.3　L命令和n text命令

为了练习 SQL*Plus 的命令，我们输入例 3-5 的 SQL 语句。

例 3-5

```
SQL> SELECT empno, ename, job, sal
  2  FROM dept
```

```
  3  WHERE sal >= 1500
  4  ORDER BY job, sal DESC;
```

例 3-5 结果

```
WHERE sal >= 1500
      *
ERROR 位于第 3 行:
ORA-00904: ????
```

例 3-5 显示的结果告诉我们，这个语句显然是错的，因为所有要显示的列都在 emp 表中而不是在 dept 表中。

毛主席说“错误总是难免的，只要改了就是好同志。”Oracle 的设计思想与毛主席他老人家的教诲是一脉相承的。也许是英雄所见略同，也许是继承了毛泽东的伟大思想，Oracle 的 SQL*Plus 提供了若干条命令来帮助我们发现错误或改正错误，其中最常用的这类命令之一就是 L（LIST）命令。该命令用来显示 SQL 缓冲区中的内容。例如，您可以使用 L（LIST）命令来显示您刚刚输入的 SQL 语句，如例 3-6 所示。

例 3-6

```
SQL> L
```

例 3-6 结果

```
  1  SELECT empno, ename, job, sal
  2  FROM dept
  3  WHERE sal >= 1500
  4* ORDER BY job, sal DESC
```

之后可以使用 n text 命令来修改出错的部分，其中，n 为在 SQL 缓冲区中的 SQL 语句的行号，text 为替代出错部分的 SQL 语句。因为从 L（LIST）命令的显示得知是第 2 行出了错，所以现在输入例 3-7 的命令来修改错误。

例 3-7

```
SQL> 2 FROM emp
```

之后，应该再使用例 3-8 的 L（LIST）命令来显示 SQL 缓冲区中的内容，以检查您的修改是否正确。

例 3-8

```
SQL> L
```

例 3-8 结果

```
  1  SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4* ORDER BY job, sal DESC
```

例 3-8 的结果表明您所做的修改准确无误。那么我们又该如何运行这条语句呢？

3.4 “/” 命令

您当然没有必要重新输入这条语句，因为这条语句已经在 SQL 缓冲区中。Oracle 提供了 SQL*Plus 命令“/”（RUN）来重新运行在 SQL 缓冲区中的 SQL 语句。于是您可以输入例 3-9 的 SQL*Plus 命令来重新运行刚刚修改过的 SQL 语句。

例 3-9

```
SQL> /
```

例 3-9 结果

```
     EMPNO ENAME                JOB                       SAL
---------- -------------------- ------------------ ----------
      7788 SCOTT                ANALYST                  3000
      7902 FORD                 ANALYST                  3000
      7566 JONES                MANAGER                  2975
      7698 BLAKE                MANAGER                  2850
      7782 CLARK                MANAGER                  2450
      7839 KING                 PRESIDENT                5000
      7499 ALLEN                SALESMAN                 1600
      7844 TURNER               SALESMAN                 1500
已选择 8 行。
```

以上的几条 SQL*Plus 命令都为我们修改程序中的错误提供了方便。当然，Oracle 提供的这类 SQL*Plus 命令远远不止这些。

3.5 n（设置当前行）命令和A[PPEND]（附加）命令

设想您输入了例 3-10 的查询语句来查询员工的信息。

例 3-10

```
SQL> SELECT ename
  2  FROM emp;
```

例 3-10 结果

```
ENAME
-------------
SMITH
ALLEN
WARD
JONES
MARTIN
```

```
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER
已选择 14 行。
```

看到以上输出时，您发现在 SELECT 子句中忘了写入 job 和 sal。这时您又如何修改您的 SELECT 子句呢？首先您应该使用 SQL*Plus 的 L（LIST）命令来显示 SQL 缓冲中的内容。

例 3-11

```
SQL> L
```

例 3-11 结果

```
  1  SELECT ename
  2* FROM emp
```

在例 3-11 显示的结果中，2 后面的“*”表示第 2 行为当前行。从例 3-11 显示的结果发现，SELECT ename 是 SQL 缓冲区中的第 1 行。为了在 ename 之后添加“,job,sal”，您应该先把第 1 行设置为当前行。于是输入 1，如例 3-12 所示，该命令把第一行置为当前行。

例 3-12

```
SQL> 1
```

例 3-12 结果

```
  1* SELECT ename
```

例 3-12 显示的结果表明已成功地将 SQL 缓冲区中的第 1 行设置为当前行。现在就可以使用例 3-13 的 a 命令（附加命令）把“,job,sal”添加到 SELECT ename 之后了。

例 3-13

```
SQL> a ,job, sal
```

例 3-13 结果

```
  1* SELECT ename,job, sal
```

此时，您应该再使用例 3-14 的 L 命令来检查您所做的修改是否正确。

例 3-14

```
SQL> L
```

例 3-14 结果

```
  1  SELECT ename,job, sal
  2* FROM emp
```

看到了例 3-14 显示的结果，您发现修改后的查询语句正是您所希望的，于是再一次输入执行命令（“/”或 R）来重新运行 SOL 缓冲区中的查询语句。这次您就可以得到您所需要的结果了，如例 3-15 所示。

例 3-15

```
SQL> /
```

例 3-15 结果

```
ENAME       JOB            SAL
---------- --------- ----------
SMITH       CLERK          800
ALLEN       SALESMAN      1600
WARD        SALESMAN      1250
JONES       MANAGER       2975
MARTIN      SALESMAN      1250
BLAKE       MANAGER       2850
CLARK       MANAGER       2450
SCOTT       ANALYST       3000
KING        PRESIDENT     5000
TURNER      SALESMAN      1500
ADAMS       CLERK         1100
JAMES       CLERK          950
FORD        ANALYST       3000
MILLER      CLERK         1300
已选择 14 行。
```

这与我们输入 SQL 语句后立即执行所得的结果完全相同。

用 n 来指定第 n 行为当前行，这里 n 为自然数。那么如果想在第 1 行之前插入一行数据，又该怎么办呢？可以使用 0 text 在第 1 行之前插入一行数据。

如果发现 SQL 缓冲区中某行的内容需要去掉，又该如何处理呢？

3.6 DEL命令

可以使用 DEL n 命令删除第 n 行。如果没有指定 n 就是删除当前行。同时也可以使用 DEL m n 命令删除从 m 行到 n 行的所有内容。为了演示如何使用这个 SQL*Plus 命令，您可以重新输入与例 3-5 几乎一样的、如例 3-16 所示的 SQL 语句。

例 3-16

```
SQL> SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4  ORDER BY job, sal DESC;
```

为了准确地确定所要删除行的行号，您可以再次使用例 3-17 的 SQL*Plus 命令。

例 3-17

```
SQL> L
```

例 3-17 结果

```
  1  SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4* ORDER BY job, sal DESC
```

假设 emp 表是一个很大的表，为了提高查询的效率，您决定去掉 ORDER BY 子句，可以使用例 3-18 的 SQL*Plus 命令来完成。

例 3-18

```
SQL> DEL 4
```

现在还是应该使用例 3-19 的 SQL*Plus 的 L 命令来检查您所做的操作是否成功。

例 3-19

```
SQL> L
```

例 3-19 结果

```
  1  SELECT empno, ename, job, sal
  2  FROM emp
  3* WHERE sal >= 1500
```

例 3-19 显示的结果表明已经成功地删除了 SQL 缓冲区中包含 ORDER BY 子句的第 4 行。此时，您可以再次使用例 3-20 的 SQL*Plus 的“/”命令运行该语句。

例 3-20

```
SQL> /
```

例 3-20 结果

```
     EMPNO ENAME                JOB                    SAL
---------- -------------------- ---------------- ---------
      7499 ALLEN                SALESMAN              1600
      7566 JONES                MANAGER               2975
      7698 BLAKE                MANAGER               2850
      7782 CLARK                MANAGER               2450
      7788 SCOTT                ANALYST               3000
      7839 KING                 PRESIDENT             5000
      7844 TURNER               SALESMAN              1500
      7902 FORD                 ANALYST               3000
已选择 8 行。
```

很显然，例 3-10 显示的结果是无序的，其易读性也下降了。但有时为了系统的整体效率，

牺牲一些查询结果的易读性也是在所难免，就像社会上常说的“牺牲小家为大家”一样。

在这里我们并没有给出 DEL m n 命令和 DEL 命令的例子，因为它们的用法与 DEL m 命令大同小异。如果读者感兴趣可以自己试一试。

除了以上所介绍的修改和删除命令之外，还有没有其他的 SQL*Plus 的命令来完成相同的操作呢？

3.7 C[HANGE]命令

也可以使用“C[HANGE]/原文/新的正文”命令来修改 SQL 缓冲区中的语句。该命令是在当前行中用“新的正文”替代“原文”。

为了演示该命令的用法，您可以重新输入与例 3-5 完全相同的 SQL 语句，如例 3-21 所示。

例 3-21

```
SQL> SELECT empno, ename, job, sal
  2  FROM dept
  3  WHERE sal >= 1500
  4  ORDER BY job, sal DESC;
```

例 3-21 结果

```
WHERE sal >= 1500
     *
ERROR 位于第 3 行:
ORA-00904: 无效列名
```

现在试着用刚刚学过的 C[HANGE]命令将 SQL 缓冲区中第 2 行的 dept 改为 emp，使用了例 3-22 的 SQL*Plus 命令。

例 3-22

```
SQL> C /dept/emp
```

例 3-22 结果

```
SP2-0023: 未找到字符串
```

例 3-22 显示的结果使您感到意外，因为您有惊人的记忆力，所以坚信您所输入的 SQL*Plus 命令没有任何错误。实际上例 3-22 的 SQL*Plus 命令是完全正确的，只是当前行不是第 2 行，即不包含 dept，所以才造成了“未找到字符串”的错误。现在您可以先输入例 3-23 的 SQL*Plus 命令将 SQL 缓冲区中第 2 行设置为当前行。

例 3-23

```
SQL> 2
```

例 3-23 结果

```
  2* FROM dept
```

然后可以重新输入与例 3-22 完全相同的 SQL*Plus 命令，如例 3-24 所示。

例 3-24

```
SQL> C /dept/emp
```

例 3-24 结果

```
  2* FROM emp
```

例 3-24 显示的结果表明您已经成功地将 SQL 缓冲区中第 2 行的 dept 修改为 emp。但为了谨慎起见，您还是应该使用例 3-25 的 L 命令来验证一下。

例 3-25

```
SQL> l
```

例 3-25 结果

```
  1  SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4* ORDER BY job, sal DESC
```

如果这时您再使用例 3-26 的“/”命令，就会得到与例 3-9 完全相同的结果。

例 3-26

```
SQL> /
```

例 3-26 结果

```
     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7788 SCOTT      ANALYST         3000
      7902 FORD       ANALYST         3000
      7566 JONES      MANAGER         2975
      7698 BLAKE      MANAGER         2850
      7782 CLARK      MANAGER         2450
      7839 KING       PRESIDENT       5000
      7499 ALLEN      SALESMAN        1600
      7844 TURNER     SALESMAN        1500
已选择 8 行。
```

如果您想使输出的结果只按工资（sal）由大到小排序，您首先应该使用例 3-27 的 SQL*Plus 命令将 SQL 缓冲区中第 4 行设置为当前行。

例 3-27

```
SQL> 4
```

例 3-27 结果

```
  4* ORDER BY job, sal DESC
```

然后就可以使用例 3-28 的 C 命令将 job 从 SQL 缓冲区第 4 行中删除。

例 3-28

```
SQL> c /job,/
```

例 3-28 结果

```
  4* ORDER BY  sal DESC
```

现在应该再使用例 3-29 的 L 命令来验证一下修改是否成功。

例 3-29

```
SQL> l
```

例 3-29 结果

```
  1  SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4* ORDER BY  sal DESC
```

最后您可以使用例 3-30 的“/”命令来运行 SQL 缓冲区中的语句。

例 3-30

```
SQL> /
```

例 3-30 结果

```
     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7839 KING       PRESIDENT       5000
      7788 SCOTT      ANALYST         3000
      7902 FORD       ANALYST         3000
      7566 JONES      MANAGER         2975
      7698 BLAKE      MANAGER         2850
      7782 CLARK      MANAGER         2450
      7499 ALLEN      SALESMAN        1600
      7844 TURNER     SALESMAN        1500
已选择 8 行。
```

从本节的讨论可以看出，在某些情况下使用 C 命令进行修改或删除操作可能比使用其他的命令更方便。

3.8　如何生成脚本文件

为了演示如何生成脚本文件，可以重新输入例 3-31 的查询语句。

例 3-31

```
SQL>  SELECT empno, ename, job, sal
   2   FROM emp
```

```
  3  WHERE sal >= 1500
  4  ORDER BY job, sal DESC;
```

例 3-31 结果

```
   EMPNO ENAME                JOB                  SAL
---------- ------------------ ----------------- ---------
    7788 SCOTT                ANALYST             3000
    7902 FORD                 ANALYST             3000
    7566 JONES                MANAGER             2975
    7698 BLAKE                MANAGER             2850
    7782 CLARK                MANAGER             2450
    7839 KING                 PRESIDENT           5000
    7499 ALLEN                SALESMAN            1600
    7844 TURNER               SALESMAN            1500
已选择 8 行。
```

现在您可以输入如例 3-32 所示的 SQL*Plus 的命令，将 SQL 缓冲区中的语句存入 D:\SQL\SAMPLE.sql 文件中。该文件也称为脚本文件。

注意：

在执行 SQL*Plus 命令之前，您要先使用操作系统命令来创建 D:\SQL 目录（文件夹）。

例 3-32

```
SQL> SAVE D:\SQL\SAMPLE
```

例 3-32 结果

```
已创建文件 D:\SQL\SAMPLE.sql
```

SAVE 命令把 SQL 缓冲区的内容存入指定的文件，该文件称为脚本文件。此时，如果您使用正文编辑器打开文件 D:\SQL\SAMPLE.sql，会在该文件中看到例 3-33 所示的内容。

例 3-33

```
SELECT empno, ename, job, sal
 FROM emp
 WHERE sal >= 1500
 ORDER BY job, sal DESC
/
```

此时如果执行 SQL*Plus 的 L 命令，您将会看到您以前输入的 SQL 语句不是 SQL*Plus 的命令 SAVE D:\SQL\SAMPLE，这说明 SQL*Plus 的命令不被存入 SQL 缓冲区。请看例 3-34。

例 3-34

```
SQL> L
```

例 3-34 结果

```
 1   SELECT empno, ename, job, sal
 2   FROM emp
 3   WHERE sal >= 1500
 4*  ORDER BY job, sal DESC
```

现在如果您输入例 3-35 的 SQL 语句，会发现在 SQL 缓冲区中有哪些变化呢？

例 3-35

```
SQL> SELECT *
  2  FROM dept;
```

例 3-35 结果

```
   DEPTNO DNAME                          LOC
---------- -------------------------- -------
       10 ACCOUNTING                   NEW YORK
       20 RESEARCH                     DALLAS
       30 SALES                        CHICAGO
       40 OPERATIONS                   BOSTON
```

之后再输入例 3-36 的 SQL*Plus 的 L 命令，您会发现 SQL 缓冲区中存的内容已变为刚刚输入的语句。

例 3-36

```
SQL> L
```

例 3-36 结果

```
 1  SELECT *
 2* FROM dept
```

以上的例子也证明了 SQL 缓冲区只能存储一个 SQL 语句。

3.9　如何编辑脚本文件

在生成了脚本文件 D:\SQL\SAMPLE.sql 之后，可以使用例 3-37 的 SQL*Plus 的 GET 命令将该脚本文件装入 SQL 缓冲区。

例 3-37

```
SQL> GET D:\SQL\SAMPLE.sql
```

例 3-37 结果

```
 1   SELECT empno, ename, job, sal
 2   FROM emp
 3   WHERE sal >= 1500
 4*  ORDER BY job, sal DESC
```

现在可以使用如例 3-38 所示的 SQL*Plus 的 L 命令来验证您是否成功地将脚本文件 D:\SQL\ SAMPLE.sql 装入了 SQL 缓冲区。

例 3-38

```
SQL> L
```

例 3-38 结果

```
  1  SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4* ORDER BY job, sal DESC
```

此时，您就可以使用前面学过的 C、A、n 或 DEL 等命令来编辑 SQL 缓冲区中的语句。也可以使用如例 3-39 所示的“/”命令来重新运行该 SQL 语句。

例 3-39

```
SQL> /
```

例 3-39 结果

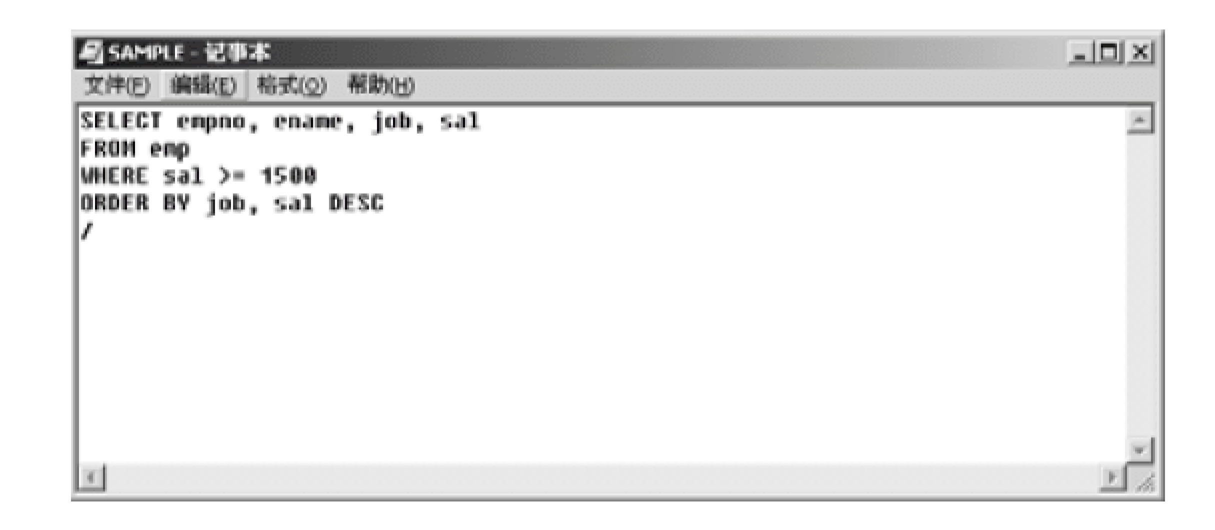
```
     EMPNO ENAME                JOB                   SAL
---------- -------------------- --------------- ---------
      7788 SCOTT                ANALYST              3000
      7902 FORD                 ANALYST              3000
      7566 JONES                MANAGER              2975
      7698 BLAKE                MANAGER              2850
      7782 CLARK                MANAGER              2450
      7839 KING                 PRESIDENT            5000
      7499 ALLEN                SALESMAN             1600
      7844 TURNER               SALESMAN             1500
已选择 8 行。
```

您也可以使用如例 3-40 所示的 SQL*Plus 的“ed[it]”命令来直接编辑 D:\SQL\SAMPLE。

例 3-40

```
SQL> ed D:\SQL\SAMPLE
```

例 3-40 结果



现在就可以在这个编辑器中对 D:\SQL\SAMPLE 进行编辑了。

3.10 如何直接运行脚本文件

也可以使用如例 3-41 所示的 SQL*Plus 命令来直接运行脚本文件 D:\SQL\SAMPLE.sql。

例 3-41

```
SQL> @D:\SQL\SAMPLE.sql
```

例 3-41 结果

```
     EMPNO ENAME                JOB                       SAL
---------- -------------------- ------------------ ----------
      7788 SCOTT                ANALYST                  3000
      7902 FORD                 ANALYST                  3000
      7566 JONES                MANAGER                  2975
      7698 BLAKE                MANAGER                  2850
      7782 CLARK                MANAGER                  2450
      7839 KING                 PRESIDENT                5000
      7499 ALLEN                SALESMAN                 1600
      7844 TURNER               SALESMAN                 1500
已选择 8 行。
```

“@”或 START 命令是把指定脚本文件的内容装入 SQL 缓冲区中并运行。

现在可以自豪地说您已经会写脚本文件了。尽管一些招聘广告上把写脚本文件说得神乎其神的，原来也不过如此。

您现在可能想问什么情况下要创建脚本文件，其原则很简单，就是如果写的 SQL 语句是将来反复使用的，您就应该把该语句装入脚本文件；如果您写的 SQL 语句只用一次，就没有必要创建脚本文件。

3.11 SPOOL命令

我们这里要介绍的最后一个 SQL*Plus 命令为 SPOOL。当要用 SQL 语句产生一个大的报表时，该命令很有用。例如，输入如例 3-42、例 3-43 和例 3-44 所示的 SQL*Plus 命令和 SQL 语句。

例 3-42

```
SQL> SPOOL D:\SQL\OUTPUT
```

例 3-43

```
SQL> SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4  ORDER BY job, sal DESC;
```

例 3-44

```
SQL> SPOOL OFF;
```

此时您可以从 D:\SQL\OUTPUT 文件中看到如下的内容。

例 3-44 结果

```
SQL> SELECT empno, ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4  ORDER BY job, sal DESC;

     EMPNO ENAME                JOB                      SAL
---------- -------------------- ------------------ --------
      7788 SCOTT                ANALYST                 3000
      7902 FORD                 ANALYST                 3000
      7566 JONES                MANAGER                 2975
      7698 BLAKE                MANAGER                 2850
      7782 CLARK                MANAGER                 2450
      7839 KING                 PRESIDENT               5000
      7499 ALLEN                SALESMAN                1600
      7844 TURNER               SALESMAN                1500
已选择 8 行。
SQL> SPOOL OFF;
```

SPOOL D:\SQL\OUTPUT 中 SPOOL 之后为文件名，该命令的含义是指在该命令之后屏幕上所显示的一切都要存到 D:\SQL 目录下的 OUTPUT 文件中。只有当输入 SPOOL OFF 之后您才能看到 OUTPUT 文件中的内容。如果您输入 SPOOL OUT 表示将其内容送到打印机。

我们已经简单地介绍了常用的 SQL*Plus 命令。如果您觉得本章的内容难懂，不要担心，因为在 Windows 上的 SQL*Plus 中，您可以使用鼠标和许多基于图形界面的编辑功能。

如果您读过其他类似的书籍，可能会发现有关这方面的内容一般都是一带而过的。我之所以用了这么大的篇幅来介绍这些令许多初学者望而生畏的命令，是因为这些命令是 SQL*Plus 的基本命令，所有操作系统上运行的 Oracle 都支持这些命令。如果读者掌握了这些命令，也就是学到了一套看家的本事，即无论遇到何种操作系统上运行的 Oracle，您都可以熟练地使用 SQL 来操作 Oracle 数据库。

提示：

如果读者在阅读 3.12~3.14 节时理解上有困难，请不要担心，因为即使不理解这些内容也不会影响以后的学习。在本书的第 1 版中并未包含这部分的内容，其实 Oracle 官方的 SQL 培训也不包括这些内容。但是在实际的工作中，很少遇到只使用 Oracle 一家公司的软件的企事业，这样就免不了要在 Oracle 系统与其他系统（软件）之间进行数据的交换，而这方面的材料比较少，也比较难理解，有时甚至很难找到。为了帮助读者能迅速地将所学到的

Oracle SQL 知识运用于实际工作中，所以在本书的新版中包括了 Oracle 与其他系统（软件）的数据交换及将这些操作的自动化等内容。

3.12 将Oracle数据库的数据导出给其他系统

如果读者登录过有关 Oracle 的论坛，可能时常会发现有一些询问其他软件怎样访问 Oracle 数据的帖子，几乎所有回帖的答案都是要先进行一些系统配置，如 ODBC、JDBC 等。这些配置工作对一般人来说并不是容易的事。那么有没有更简单的方法呢？当然有，那就是使用 SQL*Plus 的 SPOOL 命令。为了帮助读者理解，下面通过一个故事来解释其具体操作方法。

某公司现在使用 Oracle 数据库存储公司中所有的数据。但是由于历史和人为的原因，公司中软件的采购是由各个部门自己决定的，因此使用的程序设计语言和应用系统可以说是五花八门。公司决定要将 Oracle 系统与其他系统之间的数据交换自动化，于是请教了若干软件公司，他们都说要进行类似 ODBC 或 JDBC 的系统配置，还要调整数据库的配置，之后还要进行相关用户的培训等。当然收费也是相当可观，用老板的话来说“简直是狮子大开口”。

正是在这种无奈的情况下，老板找到了您这个刚进公司的见习 DBA（也是公司中唯一懂 Oracle 的员工），问您有没有办法用比较简单和经济的方法来解决困扰了公司多年的问题。于是您决定首先将一个简单而且常用的部门（dept）表中的所有数据导出给其他程序设计语言，如 C 或 Java。以下就是具体的操作步骤：

（1）为了管理方便，您首先要创建一个存放相关文件的目录（文件夹）SQL（也可以使用其他的名字）。启动资源管理器，选择“文件”→“新建”→“文件夹”命令，如图 3.1 所示。



图 3.1

（2）将新创建的文件夹名改为 SQL，如图 3.2 所示。

（3）打开记事本并写入如图 3.3 所示的命令。如果数据行很多，也可以将 pagesize 设

置得更大。在 SQL 查询语句中使用了连接字符“||”将每个字段以逗号分隔，并将导出的数据存入 E 盘下的 SQL 目录的 data.txt 正文文件，set heading off 命令是在查询语句的输出结果中不显示列名，这一点对简化使用该数据文件的软件设计和开发是相当有用的，因为读取数据的程序将不用考虑如何去掉列名之类的程序不需要的数据了（在该书的光盘中有一个名为 data.sql 的脚本文件，如果读者想节省时间，可以直接复制其中的内容，或直接将该文件复制到当前的目录中）。



图　3.2



图　3.3

（4）之后选择“文件”→“保存”命令，如图 3.4 所示。

（5）在弹出的“另存为”对话框的“保存在”下拉列表框中选择 E 盘的 SQL 目录，在“文件名”下拉列表框中输入“data.sql”，最后单击“保存”按钮，如图 3.5 所示。如果无法生成 data.sql 脚本文件，请参考本章最后的解释。



图　3.4



图　3.5

（6）启动 DOS 界面，输入“e:”命令切换到 E 盘，输入 cd sql 切换到 E:\SQL 目录，之后输入“sqlplus scott/tiger”命令进入 SQLPlus 并以 scott 用户登录数据库，如图 3.6 所示。

（7）输入“@data”并按 Enter 键运行刚创建的 SQL 脚本文件 data.sql，如图 3.7 所示。

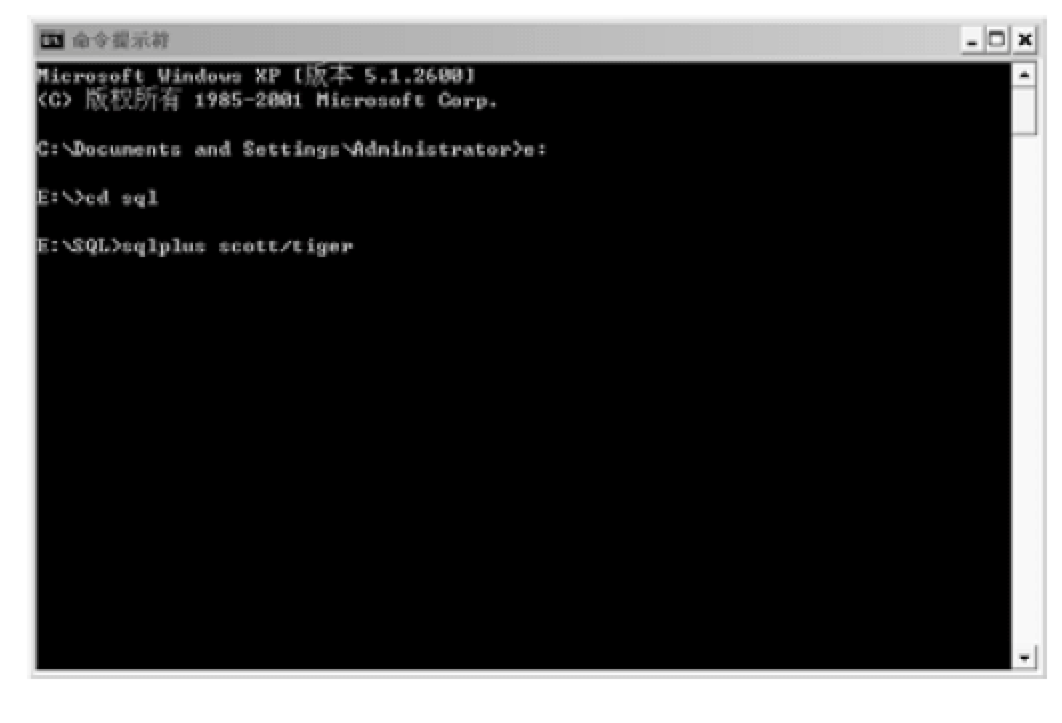


图 3.6

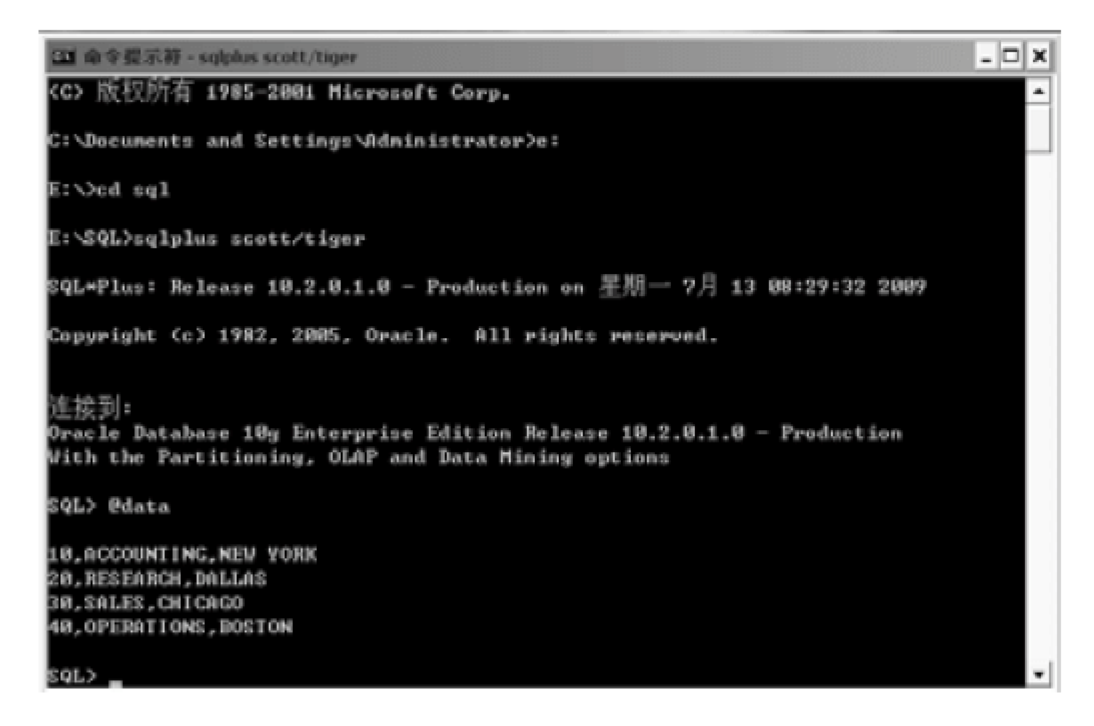


图 3.7

（8）之后打开 E:\SQL 目录，此时会发现在该目录中多了 data.txt 文本文件，这个文件就是在 data.sql 脚本文件中定义的数据文件，如图 3.8 所示。

（9）双击文件 data.txt 以打开该文件，之后将看到导出的所有数据，如图 3.9 所示。





图 3.8



图 3.9

这样，其他程序设计语言就可以操作这个文件中的数据了。分隔符不一定使用逗号，可以根据需要使用其他的符号，如分号。之后反复使用以上的方法将所需的其他表中的数据导出。现在如果有人问您会不会将 Oracle 的数据导出给其他的系统，您应该自信地回答：“会了。”

3.13 将数据导出操作自动化

当老板看到您这么快就完成了困扰公司多年的难题，可以说是喜出望外。他曾与他的秘书说：“我当时与这小子说起这件事时，只是随便说说。没想到这小子还真给做出来了。说来也怪了，当时他来应聘时表现真不怎么样，说话都吞吞吐吐的，要不是当时实在是找

不到人了，哪能要他呀！真是老天保佑，这差一点就放走个财神爷，看来是真人不露相啊！”。现在老板已经深知您的道行不浅，所以决定要进一步挖掘您的潜力。老板再次找到您让您抽空将前面的导出 Oracle 数据的操作自动化，并说如果忙不过来公司将为您配一个助手(要为一个刚进公司的见习 DBA 配助手在该公司历史上还是第 1 次）。以下就是将数据导出操作自动化的具体步骤：

（1）首先打开记事本（在 UNIX 或 Linux 系统上一般为 vi 或其他正文编辑器），然后输入操作系统命令“sqlplus /nolog @data.sql”和“exit”，如图 3.10 所示。其中，sqlplus/nolog 表示启动 sqlplus 但并不登录数据库，@data.sql 表示 SQLPlus 启动之后立即运行同一目录中的 Oracle 脚本文件 data.sql，exit 表示退出 DOS 窗口（如果没有 exit 命令，执行完所有的命令之后 DOS 窗口将留在桌面上，这显得太不专业了）。另外，在这里使用/nolog 而没有使用类似 scott/tiger 的方式启动 SQLPlus 的目的是为了安全，不让其他人看到用户的密码。



图　3.10

（2）选择“文件”→“保存”命令，弹出“另存为”对话框。在“保存在”下拉列表框中选择 E 盘的 SQL 目录（需要在操作系统上手工创建该目录，可以选择不同的盘），在“文件名”下拉列表框中输入 DownLoadData.bat（.bat 表示该文件是 DOS 操作系统的批处理文件），单击“保存”按钮，如图 3.11 所示。



图　3.11

（3）为了打开 data.sql 文件，进入 E:\SQL 目录（文件夹），用鼠标右键单击 data.sql

脚本文件，在弹出的快捷菜单中选择“编辑”命令，如图 3.12 所示。



图 3.12

（4）在第 1 行输入 SQL*Plus 命令 connect scott/tiger（其含义是以 scott 用户身份登录 Oracle 数据库）、在最后一行输入 exit 命令（如果没有 exit 命令，将把 SQL*Plus 界面留在桌面上。这样其他用户就可以清楚地看到用户的密码，同时也显得不够专业），如图 3.13 所示，然后存盘。

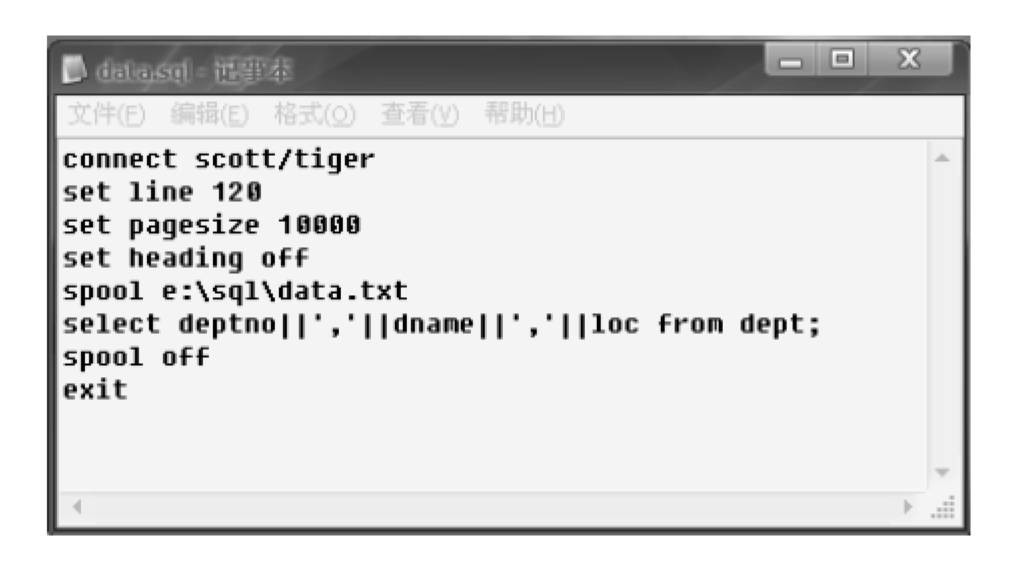


图 3.13

（5）为了演示清楚，删除已经存在的 data.txt 数据文件。使用鼠标右键单击数据文件 data.txt，在弹出的快捷菜单中选择“删除”命令就可以删除该数据文件了，如图 3.14 所示。



图 3.14

（6）用鼠标右键单击 DOS 的批处理文件 DownLoadData.bat，在弹出的快捷菜单中选择“发送到”→“桌面快捷方式”命令，将该文件的图标发送到桌面上，如图 3.15 所示。



图　3.15

（7）为了显得专业，您开始修改 DownLoadData.bat 的图标。用鼠标右键单击 DownLoadData.bat 图标，在弹出的快捷菜单中选择“属性”命令，如图 3.16 所示。弹出“DownLoadData.bat 属性”对话框。

（8）单击“更改图标”按钮，如图 3.17 所示。弹出“更改图标”对话框。



图　3.16



图　3.17

（9）选择喜欢的图标样式，然后单击“确定”按钮，如图 3.18 所示。

（10）在“DownLoadData.bat 属性”对话框中单击“应用”按钮，然后单击“确定”按钮，如图 3.19 所示。

（11）为了显得专业，还要修改 DownLoadData.bat 的文件名。用鼠标右键单击 DownLoadData.bat 的图标，在弹出的快捷菜单中选择“重命名”命令，如图 3.20 所示。

（12）然后将名字修改为看上去相当专业的“卸载 Oracle 数据”。现在双击“卸载 Oracle 数据”图标就可以完成 Oracle 数据的卸载，如图 3.21 所示。



图 3.18



图 3.19



图 3.20



图 3.21

（13）为了检验所需的数据是否成功地导出，进入 E:\SQL 目录，打开 data.txt 文件，如图 3.22 所示。检查该文件中的内容之后，您可以确信已经成功地完成了老板交给您的光荣使命。



图 3.22

3.14　商业智能软件读取Oracle数据的简单方法

当您将所有 Oracle 数据的导出工作自动化之后，老板和公司的高管们对您已经是另眼看待了，都觉得您是一个不可多得的 IT 奇才。这样您就在 IT 职业生涯的进化过程中产生了一个飞跃，从一个刚入行的“菜鸟”迅速地突变成了一名“专家”。

在一次公司高管会议上，您的顶头上司透露了他对这位新的 IT 专家的看法：“这小子真有点怪，一天我早晨上班来的早了点，大厦的保安告诉我，你们公司的灯这几天几乎每天都亮到清晨，最初保安以为公司进贼了，差一点报警，最后才发现是这小子在加班。也没人要他加班，再有咱们公司是没有加班费的，这简直不可思议！更不可思议的是，这家伙是一点眼力也没有，上班时当着我的面就打瞌睡，而且不止一次。害的我得躲着他，因为不说他，那别人也效仿怎么办？说他，他肯定不高兴。还有，这小子除了计算机和程序什么也不感兴趣，与他聊天没聊几句就扯到 IT 上，简直不会生活。只有他干活时才发现他是个难得的人才，平时看他就觉得他好像弱智一样。”

老板听了却非常高兴，并说：“以后你们这些人看到他打瞌睡时，都要躲着走。公司其实就需要这样生活上弱智的高级技术人才，以后尽量多招些这样的员工。最好所有的员工都像他一样，跟机器似的，不知疲倦地为公司卖命却不要求加工资和其他好处。公司雇了他真是太走运了，这简直跟马戏团在路边上捡到只会耍的猴子差不多。”

一天老板突然想起来，每当有许多人使用商业智能（BI）软件进行市场或其他分析与预测时公司数据库的效率就急剧下降，慢得像头牛。于是，老板又想到了您这个“弱智”的奇才，让您想想办法让系统运行得快些。您以前也没有用过商业智能软件，该公司的 Oracle 数据库也是您管理的第 1 个数据库。但是老板既然有要求也只得硬着头皮答应试一试，为了拓延时间，您与老板说您得先与相关的商业智能用户交流一下以发现究竟是什么地方的问题。老板当然答应了您的要求，并即刻通知相关的部门和人员要全力配合。

本来您以为这回可遇到麻烦了，当调查刚刚开始没多久，您就发现了幸运之神又一次眷顾了您。因为公司的商业智能用户对 Oracle 一窍不通，所以本来应该由 SQL 方便而快速完成的操作全部使用了效率极低的 BI 软件操作来完成。另外，您发现这些商业智能人员操作的是一个数据仓库系统，其数据是每隔几天更新一次。所以您决定将一些耗时的操作由 SQL 完成，并将绝大多数 BI 操作由联机改成脱机（只在数据更新后刷新相关的数据文件），当然您还要求公司增加相关 BI 人员使用的 PC 内存等。以下是将 Oracle 数据库中员工表 emp 中全部数据导出给 BI 软件的具体操作步骤：

（1）打开记事本，将如下的 SQL*Plus 命令和 SQL 语句写入记事本，如图 3.23 所示。

（2）选择“文件”→“保存”命令，弹出“另存为”对话框。在“保存在”下拉列表框中选择 E 盘的 SQL 目录，在“文件名”下拉列表框中输入“bi.sql”，最后单击“保存”按钮，如图 3.24 所示。



图 3.23



图 3.24

（3）启动 DOS 窗口，使用“e:”命令切换到 E 盘，使用 cd sql 命令进入 SQL 目录，使用“sqlplus scott/tiger”命令启动 SQLPlus 并以 scott 用户身份登录数据库，如图 3.25 所示。

（4）使用 SQLPlus 命令“@bi”来运行刚创建的 Oracle 脚本文件，如图 3.26 所示。

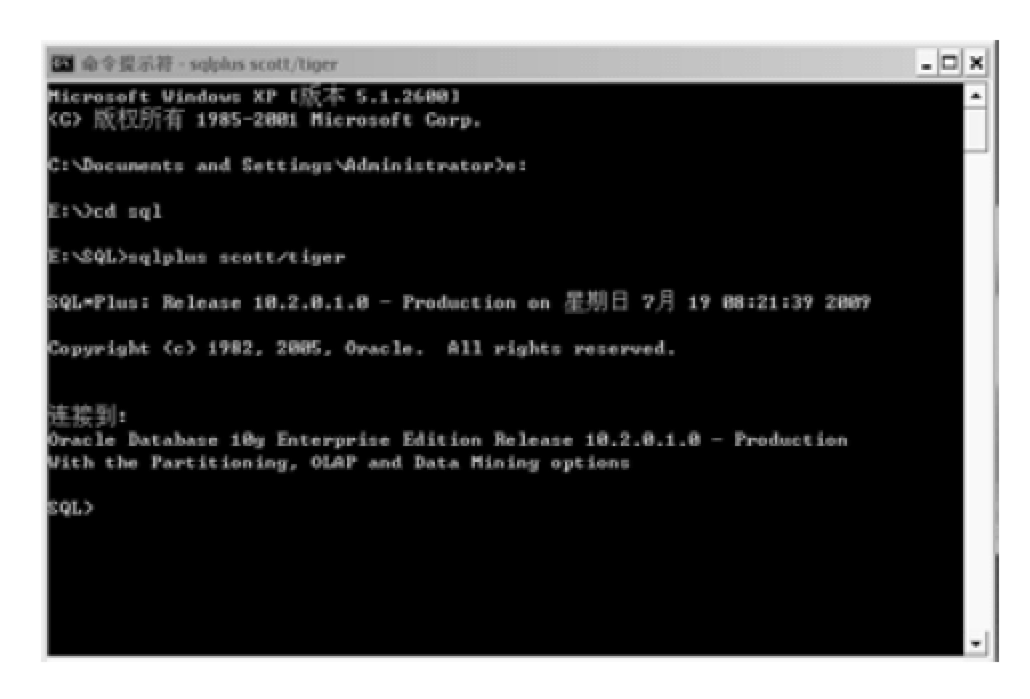


图 3.25

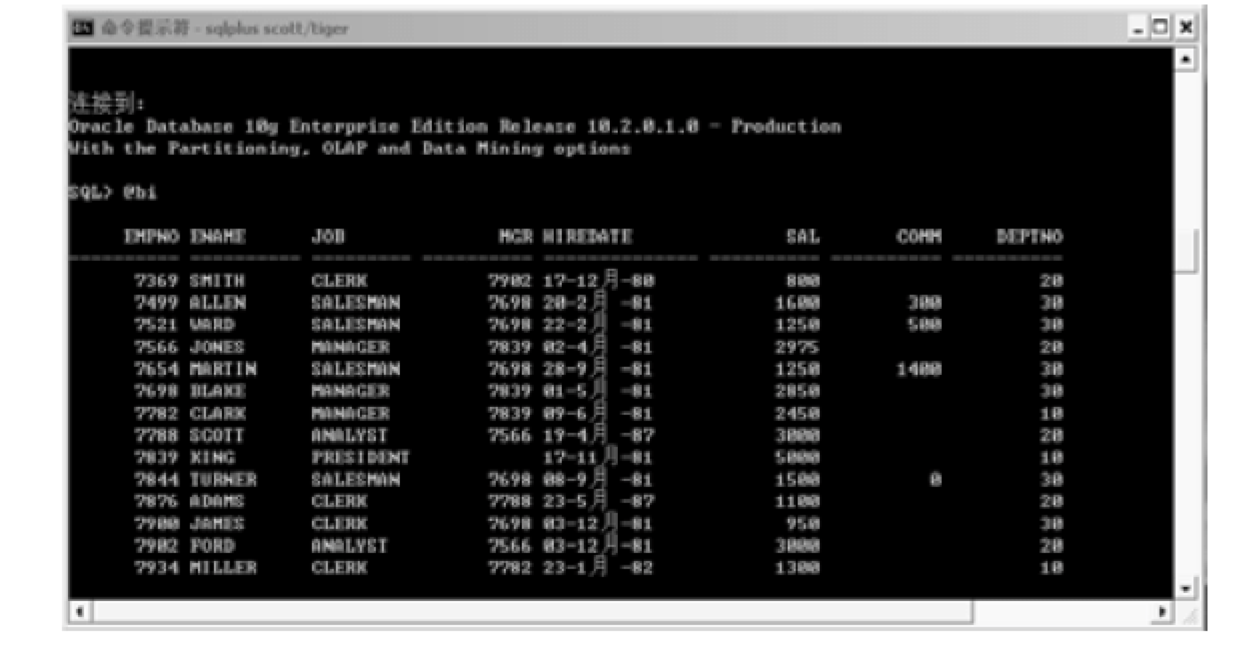


图 3.26

（5）进入 E 盘的 SQL 就可以发现刚生成的 BI_Data.txt 数据文件，如图 3.27 所示。

（6）打开 BI_Data.txt 数据文件就可以看到其中的数据，如图 3.28 所示。



图 3.27

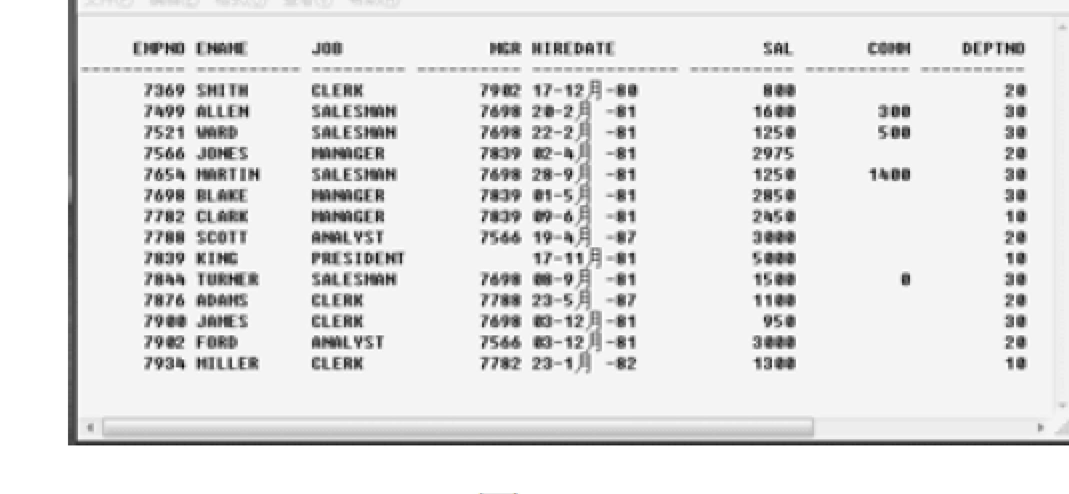


图 3.28

（7）启动 Excel，选择“文件”→“打开”命令，如图 3.29 所示。

（8）选择 E:\SQL 目录中的 BI_Data.txt 文件，如图 3.30 所示。然后，将出现 Excel 的文本导入向导界面。

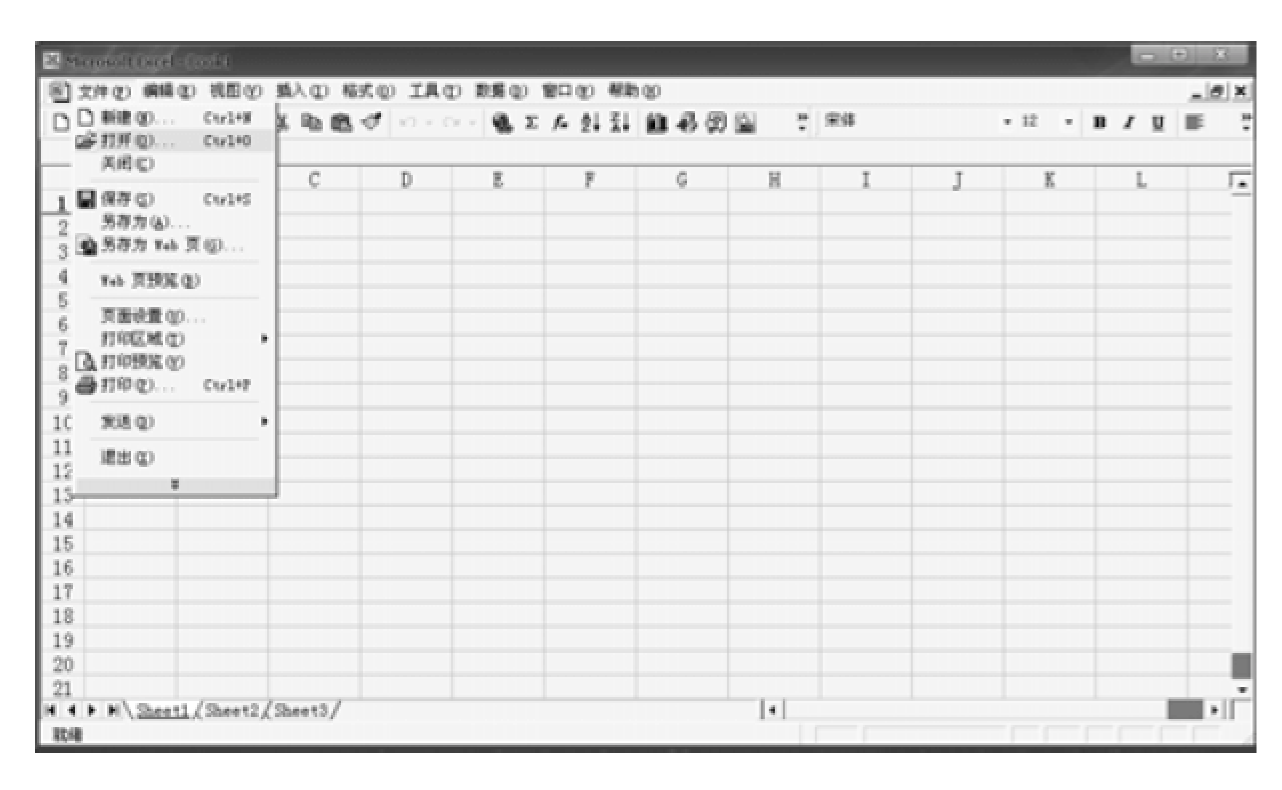


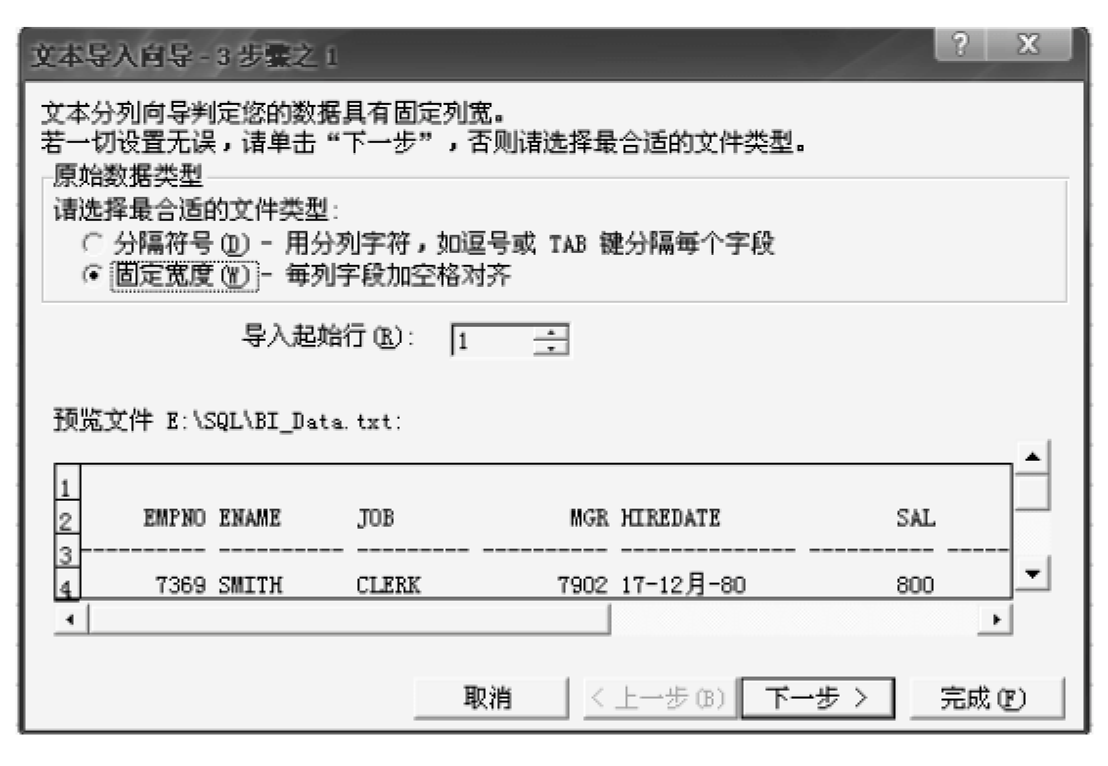
图　3.29



图　3.30

（9）在文本导入向导界面中单击“下一步”按钮，如图 3.31 所示。

（10）继续单击“下一步”按钮直到出现如图 3.32 所示的界面，最后单击“完成”按钮。



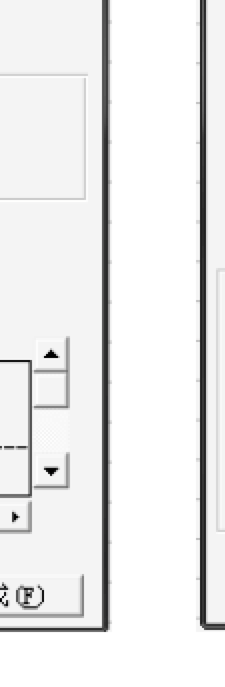


图　3.31

图　3.32

（11）然后将出现如图 3.33 所示的界面，此时 BI 人员就可以对所有的数据进行处理了。也可以选择“文件”→“保存”命令，弹出“另存为”对话框，以将这些数据存入 Excel 格式的文件。

（12）在“保存位置”下拉列表框中选择 E 盘的 SQL 目录，在“文件名”下拉列表框中输入“Excel_Data.xls”，最后单击“保存”按钮，如图 3.34 所示。



	A	B	C	D	E	F	G	H	I
1	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	
2	---------	---------	---------	---------	---------	---------	---------	----------	
3	7369	SMITH	CLERK	7902	17-12月-8	800		20	
4	7499	ALLEN	SALESMAN	7698	20-2月 -8	1600	300	30	
5	7521	WARD	SALESMAN	7698	22-2月 -8	1250	500	30	
6	7566	JONES	MANAGER	7839	02-4月 -8	2975		20	
7	7654	MARTIN	SALESMAN	7698	28-9月 -8	1250	1400	30	
8	7698	BLAKE	MANAGER	7839	01-5月 -8	2850		30	
9	7782	CLARK	MANAGER	7839	09-6月 -8	2450		10	
10	7788	SCOTT	ANALYST	7566	19-4月 -8	3000		20	
11	7839	KING	PRESIDENT		17-11月-8	5000		10	
12	7844	TURNER	SALESMAN	7698	08-9月 -8	1500	0	30	
13	7876	ADAMS	CLERK	7788	23-5月 -8	1100		20	
14	7900	JAMES	CLERK	7698	03-12月-8	950		30	
15	7902	FORD	ANALYST	7566	03-12月-8	3000		20	
16	7934	MILLER	CLERK	7782	23-1月 -8	1300		10	
17									
18									

图　3.33

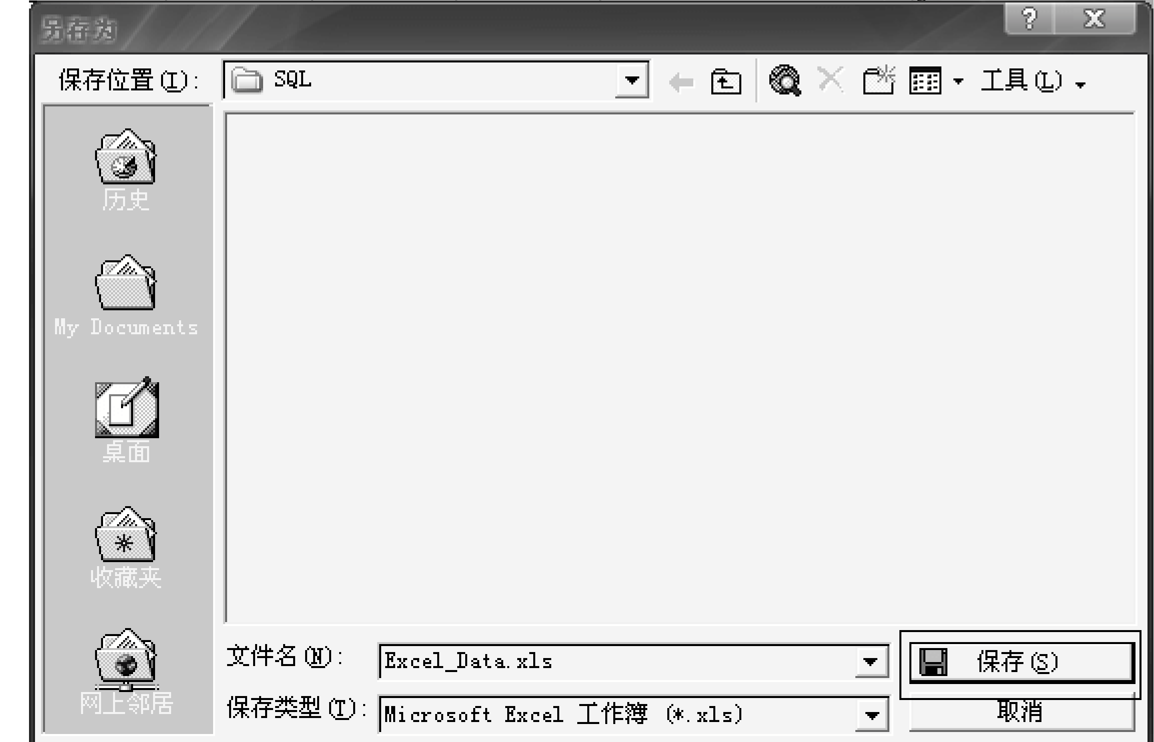


图　3.34

（13）进入 E 盘的 SQL 目录中就会发现 Excel 格式的文件 Excel_Data.xls 已经生成了，如图 3.35 所示。



图　3.35

之后 BI 人员就可以直接使用 Excel_Data.xls 这个 Excel 格式的文件，而不用每次进行一些重复的操作了。一些其他商业智能（BI）软件，如 Clementine 可以直接使用 Excel 格式的文件。

绝大多数 IT 专家们曾认为商业智能（BI）是一门伴随着 IT 而诞生的全新的学科，但是最新的考古发现却彻底颠覆了这一说法，因为考古的新发现表明 BI 这一学科在 3000 多年前已经比较成熟了，历史上最早的商业智能（BI）人员是祭司（神职人员）。

考古证据表明这些祭司们已经掌握了科学地收集信息、处理信息和快速交换信息的方法，而且祭司们可以利用信息处理的结果进行一些准确的预测。为了利益的最大化，他们编造了许多谁也看不懂的祭神仪式和经文，并谎称他/她们能通神，而将他/她们经过科学处理得到的预测说成是神的启示。当然也有不准的时候，这时就要找一个替罪羊，就要假装再与神沟通，之后说成是神说有人干了神不喜欢的事。

历史有惊人的相似之处，在科学高度发达的现代社会，虽然人们已经很少相信神之类的事。但是这些“精英”们也与时俱进，开始利用他/她们掌握的方法来套取投资者的钱财。他们在客户面前展示了一个又一个谁也看不懂、听不懂的数学模型，画出一条又一条的谁也看不懂的盈亏曲线，最后让投资者们确信只有将钱交给他/她们这些理财专家才能保证赚大钱。最终的结果是什么呢？其实大家已经知道了，就是金融海啸。

提示：

如果读者无法生成 sql 脚本文件，可能是 Windows 的设置问题。您可以按如下步骤重新设置：

（1）打开资源管理器，选择“工具”→“文件夹选项”命令，如图 3.36 所示。

（2）向下滚动滚动条，选择“查看”选项卡，取消选中“隐藏已知文件类型的扩展名”复选框，选中“在地址栏中显示完整路径”复选框，最后单击“应用”按钮，如图 3.37

所示。



图　3.36

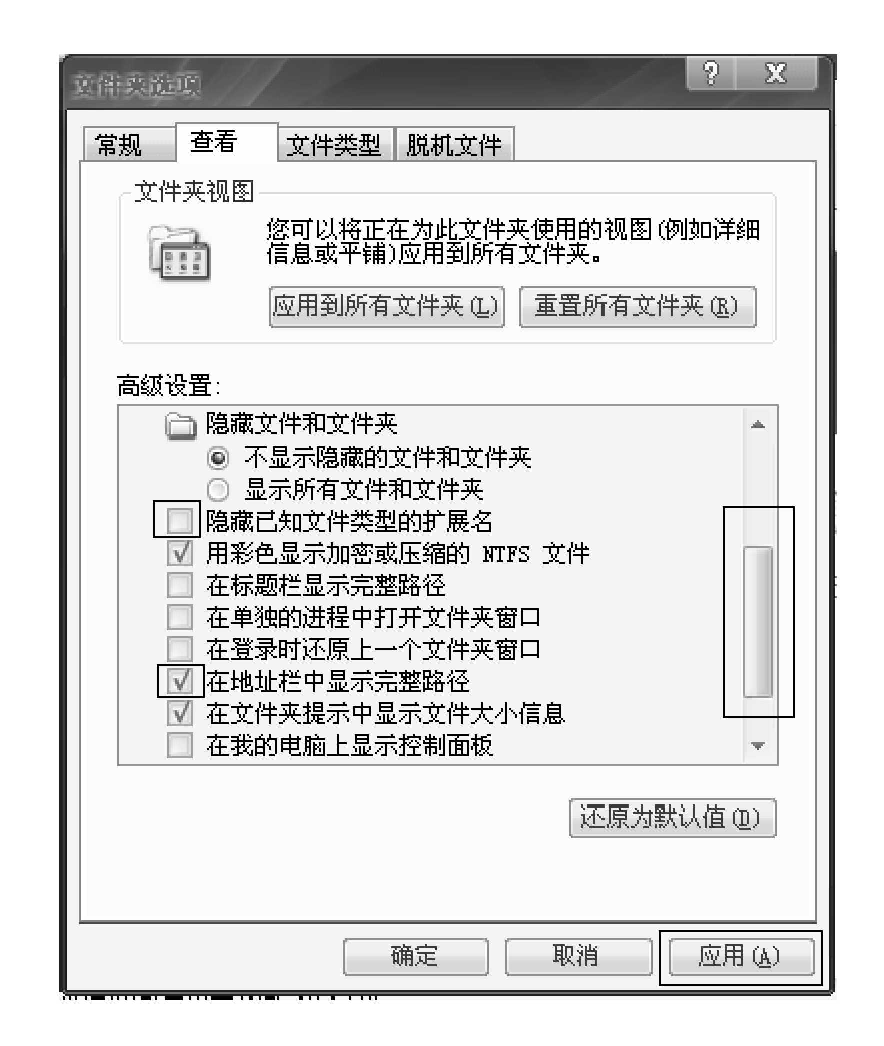


图　3.37

3.15　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 如何得到一个表的结构？
- 如何查看 SQL 缓冲区中的 SQL 语句？
- 如何修改 SQL 缓冲区中的 SQL 语句？
- 如何删除 SQL 缓冲区中的 SQL 语句？
- 如何运行 SQL 缓冲区中的 SQL 语句？
- 如何生成脚本文件？
- 如何编辑脚本文件？
- 如何直接运行脚本文件？
- 如何使用 SPOOL 命令？

第4章 单行函数

与其他的程序设计语言类似，为了方便地使用 Oracle 数据库，Oracle SQL 提供了大量的函数，实际上这些函数增强了 SQL 语言的功能。本章开始介绍 SQL 的函数。

4.1 什么是函数

什么是函数？您可以在许多书中很容易找到答案。我们在这里给出一个世俗的说明。任何东西，只要它能接收输入，对输入进行加工并产生输出，它就可以被称之为函数。

例如，牛是一个函数，它的输入是草，而产生的输出是牛奶。

函数是最受结构化程序设计者们所吹捧的一种程序设计结构。它可以有一个或多个输入但只能有一个输出，即函数只有一个出口。如果您的程序基本上都是由函数组成的话，该程序很容易调试，也很容易被重用。

4.2 单行函数简介

这一章我们所介绍的 SQL 函数都属于单行函数。单行函数的格式如下：

函数名 [(参数 1,参数 2,参数 3,…)]

其中参数可以为用户定义的常量、变量、列名和表达式。

单行函数只对表中的一行数据进行操作，并且对每一行数据只产生一个输出结果。单行函数可以接收一个或多个参数，其产生的输出结果的数据类型可能与参数的数据类型不同。

单行函数可用在 SELECT、WHERE 和 ORDER BY 的子句中，而且单行函数可以嵌套。

单行函数包含字符型、数字型、日期型、转换型和一般型函数，下面分别进行介绍。

4.3 单行字符型函数

常用的字符型函数包括 LOWER、UPPER、INITCAP、CONCAT、SUBSTR、LENGTH、INSTR、TRIM 和 REPLACE。

下面通过例子分别介绍它们。

（1）LOWER(列名|表达式)：该函数用于把字符转换成小写。

可以通过例 4-1 的查询语句来验证这一单行字符型函数。

例 4-1

```
SQL> SELECT LOWER('SQL: Structural Query Language')
  2  FROM dual;
```

例 4-1 结果

```
LOWER('SQL:STRUCTURALQUERYLANGUAGE')
------------------------------------
sql: structural query language
```

这里的 dual 是系统的一个虚表（伪表）。Oracle 系统为什么要引入这个虚表呢？

我们在第 1 章中介绍过，在查询语句中必须包含 SELECT 和 FROM 两个子句。

可是 LOWER('SQL: Structural Query Language')不属于任何表，如何在不违反 SQL 的语法前提下显示它呢？Oracle 提供的虚表 dual 就是用来解决这一难题的。

（2）UPPER(列名|表达式)：该函数用于把字符转换成大写。

可以通过例 4-2 的查询语句来验证这一单行字符型函数。

例 4-2

```
SQL> SELECT UPPER ('sql is used exclusively in rdbmses')
  2  FROM dual;
```

例 4-2 结果

```
UPPER('SQLISUSEDEXCLUSIVELYINRDBMSES')
--------------------------------------
SQL IS USED EXCLUSIVELY IN RDBMSES
```

（3）INITCAP(列名|表达式)：该函数用于把每个字的头一个字符转换成大写，其余的转换成小写。

可以通过如例 4-3 的查询语句来验证这一单行字符型函数。

例 4-3

```
SQL> SELECT INITCAP('SQL is an ENGLISH LIKE language')
  2  FROM dual;
```

例 4-3 结果

```
INITCAP('SQLISANENGLISHLIKELANGUAGE')
-------------------------------------
Sql Is An English Like Language
```

（4）CONCAT(列名|表达式,列名|表达式)：该函数用于把第 1 个字符串和第 2 个字符串连接成一个字符串。

可以通过例 4-4 的查询语句来验证这一单行字符型函数。

例 4-4

```
SQL> SELECT CONCAT('SQL allows you to manipulate the data in DB',
  2                ' without any programming knowledge')
  3  FROM dual;
```

例 4-4 结果

```
CONCAT('SQLALLOWSYOUTOMANIPULATETHEDATAINDB','WITHOUTANYPROGRAMMINGKNOW
LEDGE')
-----------------------------------------------------------------------
SQL allows you to manipulate the data in DB without any programming knowledge
```

（5）SUBSTR(列名|表达式,m,[n])：该函数用于返回指定的子串，该子串从第 m 个字符开始，其长度为 n。

可以通过例 4-5 的查询语句来验证这一单行字符型函数。

例 4-5

```
SQL> SELECT SUBSTR('SQL lets you concentrate on what has to be done',14)
  2  FROM dual;
```

例 4-5 结果

```
SUBSTR('SQLLETSYOUCONCENTRATEONWHATHASTOBEDONE',14)
---------------------------------------------------
concentrate on what has to be done
```

在这个例子中我们省略了 n，其结果是返回从第 14 个字符开始一直到结尾的所有字符。

（6）LENGTH(列名|表达式)：该函数用于返回列或表达式中字符串的长度。

可以通过例 4-6 的查询语句来验证这一单行字符型函数。

例 4-6

```
SQL> SELECT LENGTH('SQL does not let you concentrate on how it will be achieved')
  2  FROM dual;
```

例 4-6 结果

```
LENGTH('SQLDOESNOTLETYOUCONCENTRATEONHOWITWILLBEACHIVED')
---------------------------------------------------------
                                                      58
```

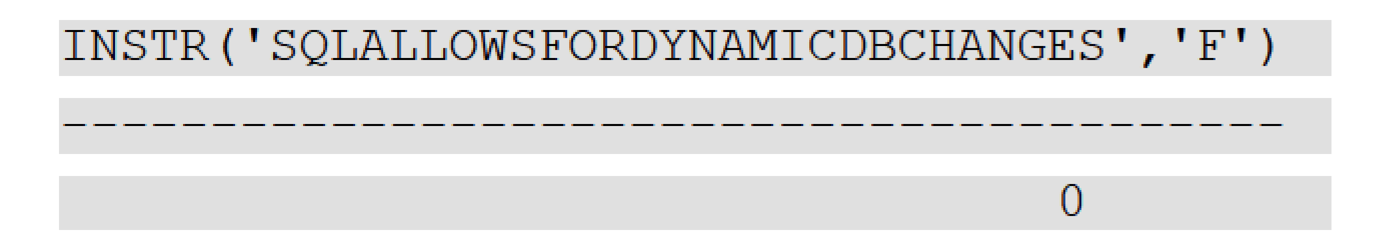
（7）INSTR(列名|表达式,'字符串',[m],[n])：该函数用于返回所给字符串的数字位置，m 表示从第 m 个字符开始搜索，n 表示所给字符串出现的次数，它们的默认值都为 1。

您可以通过例 4-7 的查询语句来验证这一单行字符型函数。

例 4-7

```
SQL> SELECT INSTR('SQL allows for dynamic DB changes', 'F')
  2  FROM dual;
```

例 4-7 结果



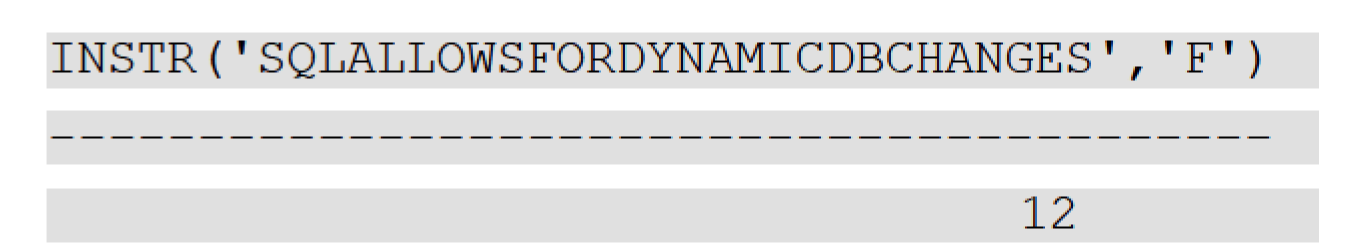
```
INSTR('SQLALLOWSFORDYNAMICDBCHANGES','F')
-----------------------------------------
                                        0
```

上面的例子说明所给的字符串是与大小写有关的，因此，我们只要把例 4-7 中的 F 改为 f 就可以得到例 4-8 所需的结果。

例 4-8

```
SQL> SELECT INSTR('SQL allows for dynamic DB changes', 'f')
  2  FROM dual;
```

例 4-8 结果



```
INSTR('SQLALLOWSFORDYNAMICDBCHANGES','F')
-----------------------------------------
                                       12
```

如果读者有时间的话，请仔细阅读例 4-1~例 4-8 中的字符串，它们实际上是一个有关 SQL 的说明。

为了帮助读者理解，在这里给出如下中文译文。

- 例 4-1：SQL 结构化的查询语言。
- 例 4-2：SQL 仅适用于关系型数据库管理系统。
- 例 4-3：SQL 是一种与英语相似的语言。
- 例 4-4：SQL 允许您在不具备任何程序设计知识的情况下操纵数据库中的数据。
- 例 4-5：SQL 让您把精力集中在必须要做的事情上。
- 例 4-6：SQL 不让您把精力集中在您如何去做这件事情上。
- 例 4-7：SQL 允许动态地改变数据库。
- 例 4-8：SQL 允许动态地改变数据库。

当您仔细地阅读完以上有关 SQL 的说明之后，相信您对 SQL 的理解一定会更加深刻，也许会达到一个新的层次。

（8）TRIM([leading|trailing|both]要去掉的字符 FROM 源字符串)：该函数能够从“源字符串”中的头（leading）部、尾（trailing）部或头部和尾部（both）中去掉“要去掉的字符”。

如果没有指定头或尾，TRIM 函数按默认（both）处理（该函数是 8i 刚引入的，在 8i 之前的版本中是 LTRIM 和 RTRIM 两个函数）。

可以通过例 4-9 的查询语句来去掉 SQL*Plus 前面的问号。

例 4-9

```
SQL> SELECT TRIM('?' FROM '?SQL*PLUS is the SQL implementation
  2               used in an Oracle RDBMS or ORDBMS.')
  3  FROM dual;
```

例 4-9 结果

```
TRIM('?'FROM'?SQL*PLUSISTHESQLIMPLEMENTATIONUSEDINANORACLERDBMSORORDBMS.')
----------------------------------------------------------------------
SQL*PLUS is the SQL implementation
           used in an Oracle RDBMS or ORDBMS.
```

也可以通过例 4-10 的查询语句来去掉 rows 之后的两个问号。

例 4-10

```
SQL> SELECT TRIM('?'  FROM 'It can process data in sets of rows??')
  2  FROM dual;
```

例 4-10 结果

```
TRIM('?'FROM'ITCANPROCESSDATAINSETSOFROWS??')
---------------------------------------------
It can process data in sets of rows
```

以上两个例子都没有指出是去掉头部的还是尾部的字符，因此 Oracle 按默认 both 来处理。但在例 4-9 中只在头部有“?”；而例 4-10 中只在尾部有“?”，所以 Oracle 系统可以正确地处理。但在例 4-11 中，TRIM 函数的这种用法就遇到麻烦了。

例 4-11

```
SQL> SELECT TRIM('s' FROM 'sql*plus is a fourth generation query languages')
  2  FROM dual;
```

例 4-11 结果

```
TRIM('S'FROM'SQL*PLUSISAFOURTHGENERATIONQUERYLANGUAGES')
--------------------------------------------------------
ql*plus is a fourth generation query language
```

在例 4-11 的 SQL 语句中，我们是想去掉尾部的 s，但 Oracle 系统把开头的和结尾的 s 都去掉了，这显然不是我们所希望的结果。这时可以用 trailing 选项来解决这个问题，如例 4-12 的查询语句。

例 4-12

```
SQL> SELECT TRIM(trailing 's' FROM 'sql*plus is a fourth generation query
languages')
  2  FROM dual;
```

例 4-12 结果

```
TRIM(TRAILING'S'FROM'SQL*PLUSISAFOURTHGENERATIONQUERYLANGUAGES')
----------------------------------------------------------------
sql*plus is a fourth generation query language
```

这次终于得到了所需的结果。

我们并没有给出在 TRIM 函数中直接使用 Leading 选项的例子。如果读者感兴趣的话，

不妨自己试着构造一个或几个这样的例子。

（9）REPLACE(正文表达式,要搜寻的字符串,替换字符串)：该函数用于在“正文表达式”中查找“要搜寻的字符串”，如果找到了就用“替换字符串”替代。

可以通过例 4-13 的查询语句来验证这一单行字符型函数。

例 4-13

```
SQL> SELECT REPLACE('SQL*PLUS supports loops or if statements', 'supports',
  2                'does not support')
  3  FROM dual;
```

例 4-13 结果

```
REPLACE('SQL*PLUSSUPPORTSLOOPSORIFSTATEMENTS','SUPPORTS','DOESNOTSUPPORT')
-----------------------------------------------------------------------
SQL*PLUS does not support loops or if statements
```

如果读者有时间的话，请仔细阅读例 4-9~例 4-13 中的字符串，它们实际上是对 SQL* Plus 的说明。

为了帮助读者理解，在这里给出如下的中文译文。

- 例 4-9：SQL*Plus 是 SQL 的一种实现，它用在 Oracle 的关系型数据库管理系统或面向对象的关系型数据库管理系统中。
- 例 4-10：它可以一次处理多行数据。
- 例 4-11：SQL*Plus 是一种第四代的查询语言。
- 例 4-12：SQL*Plus 是一种第四代的查询语言。
- 例 4-13：SQL*Plus 不支持循环和判断（分支）语句。

当仔细地阅读完以上有关 SQL*Plus 的说明之后，相信您不但会对 SQL*Plus 的理解更加深刻，而且对 SQL 和 SQL*Plus 之间的关系也会更加清楚，也许会达到一个更高的层次。

4.4　使用单行字符型函数的实例

现在我们考虑一个现实点的例子。您还记得第 2 章中的例 2-6 吗？在那个例子中我们给出了例 4-14 的查询语句。

例 4-14

```
SQL> SELECT empno, ename, job
  2  FROM emp
  3  WHERE JOB = 'salesman';
```

例 4-14 结果

```
未选定行
```

这一查询是没有结果的。在例 2-7 中把字符串中的所有字符都改成了大写后得到了正确的结果，但对一个实际的系统来说未必那么简单。如果有的记录中的 Job 一列的值为

Salesman，又该如何处理呢？可以使用例 4-15 的 SQL 语句。

例 4-15

```
SQL> SELECT empno AS "Code", UPPER(ename) NAME, INITCAP(job) "Job"
  2  FROM emp
  3  WHERE LOWER(job) = 'salesman';
```

例 4-15 结果

```
      Code NAME                 Job
---------- -------------------- ---------
      7499 ALLEN                 Salesman
      7521 WARD                 Salesman
      7654 MARTIN                Salesman
      7844 TURNER                Salesman
```

在例 4-15 的查询语句中只有最后一行的修改——LOWER(job)是必要的，其他的修改只是为了使输出显示得更清楚一些。也可以将最后一行修改为：

```
WHERE UPPER(job)='SALESMAN';
```

您也可以使用例 4-16 的语句来练习一下本节介绍过的其他函数。

例 4-16

```
SQL> SELECT CONCAT(ename, job) "Employee", SUBSTR(job,1,5) "Title",
  2            LENGTH(ename) "Length",INSTR(job, 'M')
  3  FROM emp
  4  WHERE LOWER(job) = 'salesman';
```

例 4-16 结果

```
Employee                       Title          Length INSTR(JOB,'M')
------------------------------ ---------- ---------- --------------
ALLENSALESMAN                   SALES              5              6
WARDSALESMAN                    SALES              4              6
MARTINSALESMAN                  SALES              6              6
TURNERSALESMAN                  SALES              6              6
```

我们已经介绍了字符型函数，下面开始介绍数字型函数。

4.5　数字型函数

常用的数字型函数包括 ROUND、TRUNC 和 MOD。

- ROUND(列名|表达式, n)：该函数将列名或表达式所表示的数值四舍五入到小数点后的 n 位。
- TRUNC(列名|表达式, n)：该函数将列名或表达式所表示的数值截取到小数点后的

n 位。

- MOD(m, n)：该函数将 m 除以 n 并取余数。

以上 3 个函数不再给出更详细的文字描述，下面通过例子来解释它们。我个人认为例子要比文字描述更容易理解，例如，文学作品中在描述一位美人时可能用了半页甚至一页纸来描述她如何的美丽，有时还会用上“国色天香”、“倾国倾城”等顶级的词汇来形容。但对一个正常的人来说还是很难想象出这位美人的容貌，如果看一张她的画或照片，立刻就可以知道她的美貌了。

例 4-17 的查询语句是利用 ROUND 和 TRUNC 函数使所得的结果精确到小数点后一位。

例 4-17

```
SQL> SELECT ROUND(168.888,1), TRUNC(168.888,1)
  2  FROM dual;
```

例 4-17 结果

```
ROUND(168.888,1) TRUNC(168.888,1)
---------------- ----------------
           168.9            168.8
```

从例 4-17 显示的结果，您应该看出了 ROUND 函数和 TRUNC 函数的区别。

例 4-18 的查询语句是利用 ROUND 和 TRUNC 函数使所得的结果精确到小数点后两位。

例 4-18

```
SQL> SELECT ROUND(168.333,2), TRUNC(168.333,2)
  2  FROM dual;
```

例 4-18 结果

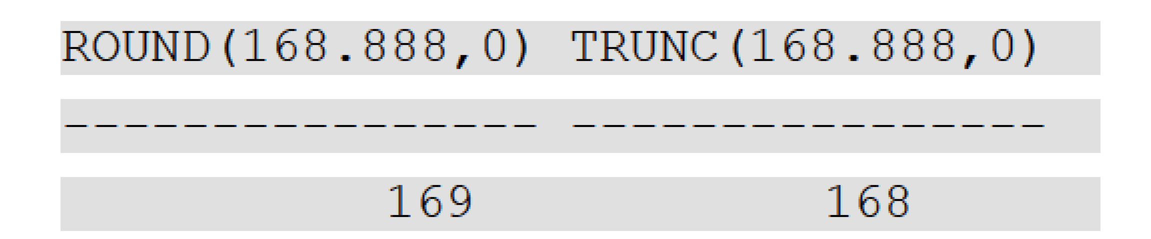
```
ROUND(168.333,2) TRUNC(168.333,2)
---------------- ----------------
          168.33           168.33
```

例 4-19 的查询语句是利用 ROUND 和 TRUNC 函数使所得的结果精确到个位。

例 4-19

```
SQL> SELECT ROUND(168.888,0), TRUNC(168.888,0)
  2  FROM dual;
```

例 4-19 结果



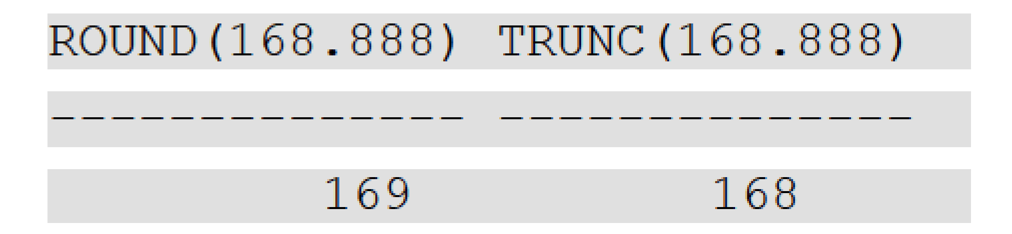
```
ROUND(168.888,0) TRUNC(168.888,0)
---------------- ----------------
             169              168
```

例 4-20 的查询语句也是利用 ROUND 和 TRUNC 函数使所得的结果精确到个位。

例 4-20

```
SQL> SELECT ROUND(168.888), TRUNC(168.888)
  2  FROM dual;
```

例 4-20 结果

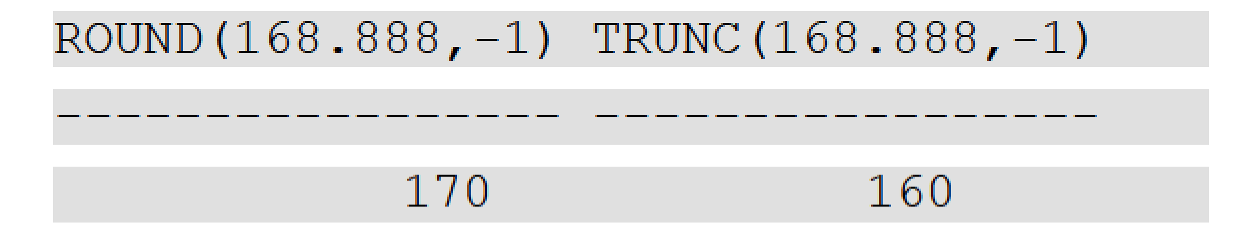

```
ROUND(168.888) TRUNC(168.888)
-------------- --------------
           169            168
```

例 4-21 的查询语句是利用 ROUND 和 TRUNC 函数使所得的结果精确到十位。

例 4-21

```
SQL> SELECT ROUND(168.888, -1), TRUNC(168.888, -1)
  2  FROM dual;
```

例 4-21 结果

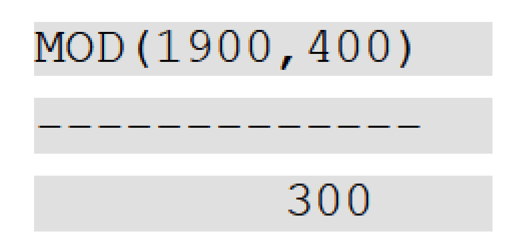

```
ROUND(168.888,-1) TRUNC(168.888,-1)
----------------- -----------------
              170               160
```

例 4-22 的查询语句是利用 MOD 函数求 1900 除以 400 后的余数。

例 4-22

```
SQL> SELECT MOD(1900, 400)
  2  FROM dual;
```

例 4-22 结果

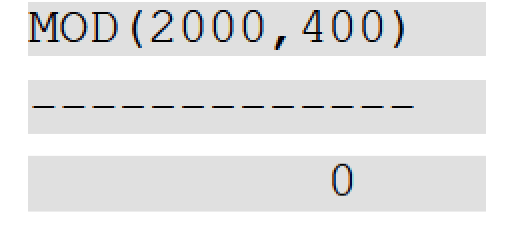

```
MOD(1900,400)
-------------
          300
```

例 4-23 的查询语句是利用 MOD 函数求 2000 除以 400 后的余数。

例 4-23

```
SQL> SELECT MOD(2000, 400)
  2  FROM dual;
```

例 4-23 结果


```
MOD(2000,400)
-------------
            0
```

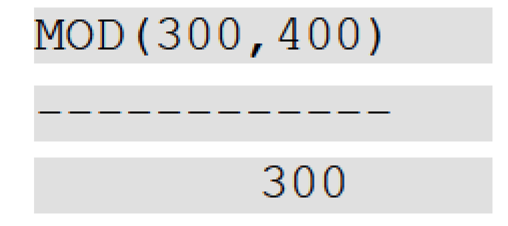
因为 2000 能被 400 除尽，所以余数为 0。

例 4-24 的查询语句是利用 MOD 函数求 300 除以 400 后的余数。

例 4-24

```
SQL> SELECT MOD(300, 400)
  2  FROM dual;
```

例 4-24 结果


```
MOD(300,400)
------------
         300
```

因为 300 比 400 小，不够除，所以余数就为 300。应该记住 Oracle 的这一规定。

您也许认为用四舍五入（ROUND）或舍去（TRUNC）函数就差那么一点，为什么还要那么计较呢？但是要注意，很多大公司财务报表是以万为单位的，而政府的财政报表或预算可能是以亿为单位的。在这种情况下什么时候用 ROUND、什么时候用 TRUNC 就不是那么简单的事了。如果监管部门没有明确规定的话，您可以巧妙地运用这两个函数为您的公司节省不少的钱（很可能比您的工资要高得多）而并不违反法律（这也可能是同时引入这两个函数的原因吧）。

4.6　日期型数据的处理

Oracle 的日期型数据的内部存储格式为：世纪，年，月，日，时，分，秒。不论您的输入格式如何，Oracle 永远按它的内部存储格式来存储日期型数据，这也许是因为日期对商业决策乃至人类社会太重要了，几乎所有的重大决策都要在指定的时间之内作出，事后诸葛亮是没用的。

从 Oracle 9i 开始，日期型数据输入和输出的默认格式为 DD-MON-RR，而在之前的版本中为 DD-MON-YY（如果对它们感到有些困惑的话，请不用担心，因为在以后章节还要介绍），这大概是为了绕开“2000 年问题”。

您可以输入 Oracle 的有效日期是从公元前 4712 年 1 月 1 日到公元 9999 年 12 月 31 日，您也许认为这是没有必要的。设想一种极端的可能，再过了几百年或几千年的某一时刻，人类遭到了灭顶之灾，这次大灾难之后只有少数偏远地区的人幸存下来了，他们又开始重建人类的文明，又过了几百年或几千年，有人发现了您今天为公司做的一份报表而且上面记载着准确的时间。我们可以想象，当时人们如何地欣喜若狂。这个考古学的重大发现如同一次巨大的地震一般，它可能要改写当时的考古学。不但发现您的报表的人会成为全球最伟大的考古学家，而且您的这份报表也会成为千百个考古学家和人类学家及博士们所引用的最权威的文献。

虽然梦想可以是美好和轻松的，但它毕竟是梦，我们现在还是不得不继续我们有关日期型数据的讨论。如果您使用的是中文操作系统，而数据库的字符集为美国英语，为了使日期型数据的显示正确，您应该输入例 4-25 的 SQL 命令。

例 4-25

```
SQL> alter session set NLS_DATE_LANGUAGE = 'AMERICAN';
```

例 4-25 结果

```
会话已更改。
```

同其他的系统一样，Oracle 也提供了系统日期函数 SYSDATE，它返回当前的系统时间，可以使用例 4-26 的查询语句来得到当前的系统时间。

例 4-26

```
SQL> SELECT SYSDATE
  2  FROM dual;
```

例 4-26 结果

```
SYSDATE
---------
05-MAY-02
```

您可以把一个日期型数据和一个数字相减，其结果仍为日期型，可以使用例 4-27 的查询语句来进行验证。

例 4-27

```
SQL> SELECT SYSDATE - 10
    2  FROM dual;
```

例 4-27 结果

```
SYSDATE-1
---------
25-APR-02
```

因为在做这个例子时，系统的当前日期为 2002 年 5 月 5 日，所以当前日期减去 10 天为 2002 年 4 月 25 日。在做这个例子时，您应该得到不同的日期。

您也可以把一个日期型数据和一个数字相加，其结果仍为日期型，可以使用例 4-28 的查询语句来进行验证。

例 4-28

```
SQL> SELECT SYSDATE + 10
    2  FROM dual;
```

例 4-28 结果

```
SYSDATE+1
---------
15-MAY-02
```

可以把两个日期型数据相减，其结果为数字型，可以使用例 4-29 的查询语句来进行验证。

例 4-29

```
SQL> SELECT TO_DATE('15-JUL-02') - SYSDATE
    2  FROM dual;
```

例 4-29 结果

```
TO_DATE('15-JUL-02')-SYSDATE
----------------------------
                   70.373669
```

如果您的操作系统和数据库都是中文系统的话，也可以将例 4-29 的查询语句改为例 4-30 的 SQL 语句。

例 4-30

```
SQL> SELECT TO_DATE('15-5 月-03') - SYSDATE
  2  FROM dual;
```

例 4-30 结果

```
TO_DATE('15-5 月-03')-SYSDATE
-----------------------------
                   6.35358796
```

这里的 TO_DATE 为一个 Oracle 提供的函数，它用于把字符串转换成日期型数据。

您也可以把一个日期型数据和一个小时数相加减后除以 24，其结果仍为日期型，可以使用例 4-31 和例 4-32 的查询语句来进行验证。

例 4-31

```
SQL> SELECT SYSDATE - 22/24
  2  FROM dual;
```

例 4-31 结果

```
SYSDATE-2
---------
04-MAY-02
```

例 4-32

```
SQL> SELECT SYSDATE + 22/24
  2  FROM dual;
```

例 4-32 结果

```
SYSDATE+2
---------
06-MAY-02
```

您也可以使用例 4-33 的查询语句来得到所有推销员的工龄。

例 4-33

```
SQL> SELECT empno, ename, job, sal, (SYSDATE-hiredate)/365 "Years"
  2  FROM emp
  3  WHERE job LIKE 'SAL%';
```

例 4-33 结果

```
     EMPNO ENAME                JOB                       SAL      Years
---------- -------------------- ------------------ ---------- ----------
      7499 ALLEN                SALESMAN                 1600 21.2182074
      7521 WARD                 SALESMAN                 1250  21.212728
      7654 MARTIN               SALESMAN                 1250 20.6154677
      7844 TURNER               SALESMAN                 1500 20.6702622
```

4.7 日 期 函 数

常用的日期型函数包括 MONTHS_BETWEEN、ADD_MONTHS、NEXT_DAY 和 LAST_DAY。

下面通过例子来分别介绍这些日期型函数。

1．MONTHS_BETWEEN(日期 1，日期 2)

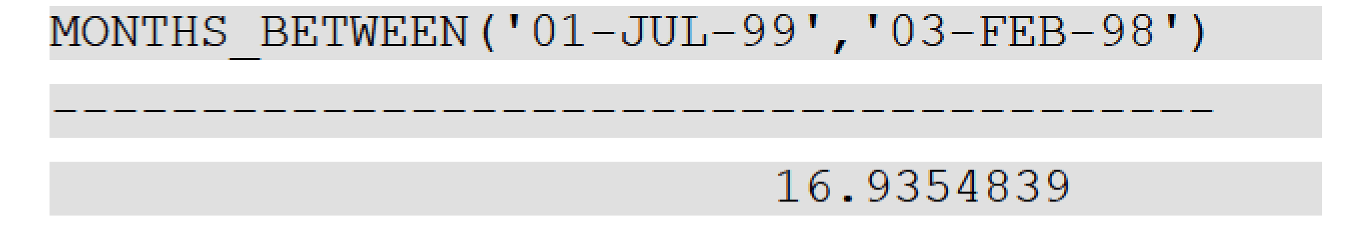
该函数用于返回日期 1 和日期 2 之间的月数。如果日期 1 大于日期 2，其返回的月数为正；如果日期 1 小于日期 2，其返回的月数为负。

可以通过例 4-34 的查询语句来验证这一单行日期型函数。

例 4-34

```
SQL> SELECT MONTHS_BETWEEN('01-JUL-99','03-FEB-98')
  2  FROM dual;
```

例 4-34 结果



```
MONTHS_BETWEEN('01-JUL-99','03-FEB-98')
---------------------------------------
                             16.9354839
```

例 4-34 显示的结果表示，1999 年 7 月 1 日和 1998 年 2 月 3 日之间相差 16.9354839 个月。

2．ADD_MONTHS(日期,n)

该函数用于把 n 个月加到日期上。

可以通过例 4-35 的查询语句验证这一单行日期型函数。

例 4-35

```
SQL> SELECT ADD_MONTHS('15-OCT-01', 8)
  2  FROM dual;
```

例 4-35 结果

```
ADD_MONTH
---------
15-JUN-02
```

例 4-35 显示的结果表示，2001 年 10 月 15 日再过 8 个月为 2002 年 6 月 15 日。

3．NEXT_DAY(日期,字符串)

该函数用于返回下一个由字符串（星期几）指定的日期。

可以通过例 4-36 的查询语句来验证这一单行日期型函数。

例 4-36

```
SQL> SELECT NEXT_DAY('10-MAY-02','MONDAY')
  2  FROM dual;
```

例 4-36 结果

```
NEXT_DAY(
---------
13-MAY-02
```

例 4-36 显示的结果表示：从 2002 年 5 月 10 日开始的下一个星期一是 2002 年 5 月 13 日。

4．LAST_DAY(日期)

该函数用于返回该日期所在月的最后一天。
可以通过例 4-37 的查询语句来验证这一单行日期型函数。

例 4-37

```
SQL> SELECT LAST_DAY('08-FEB-02')
  2  FROM dual;
```

例 4-37 结果

```
LAST_DAY(
---------
28-FEB-02
```

例 4-38 为一个综合的例子。

例 4-38

```
SQL> SELECT ename, hiredate, LAST_DAY(hiredate), NEXT_DAY(hiredate, 'SUNDAY'),
  2        MONTHS_BETWEEN(SYSDATE, hiredate) "Months",
  3        ADD_MONTHS(hiredate, 3) "Review"
  4  FROM emp;
```

例 4-38 结果

```
ENAME                HIREDATE  LAST_DAY( NEXT_DAY(     Months Review
-------------------- --------- --------- --------- ---------- ---------
SMITH                17-DEC-80 31-DEC-80 21-DEC-80 256.656332 17-MAR-81
ALLEN                20-FEB-81 28-FEB-81 22-FEB-81 254.559558 20-MAY-81
WARD                 22-FEB-81 28-FEB-81 01-MAR-81 254.495042 22-MAY-81
JONES                02-APR-81 30-APR-81 05-APR-81 253.140203 02-JUL-81
MARTIN               28-SEP-81 30-SEP-81 04-OCT-81 247.301493 28-DEC-81
BLAKE                01-MAY-81 31-MAY-81 03-MAY-81 252.172461 01-AUG-81
CLARK                09-JUN-81 30-JUN-81 14-JUN-81 250.914397 09-SEP-81
SCOTT                19-APR-87 30-APR-87 26-APR-87 180.591816 19-JUL-87
KING                 17-NOV-81 30-NOV-81 22-NOV-81 245.656332 17-FEB-82
TURNER               08-SEP-81 30-SEP-81 13-SEP-81 247.946655 08-DEC-81
```

```
ADAMS              23-MAY-87 31-MAY-87 24-MAY-87 179.462784 23-AUG-87
…
```

如果您的操作系统和数据库都是中文系统，则应该将例 4-38 中第 1 行的末尾处的 SUNDAY 改为星期日，其 SQL 语句如例 4-39。

例 4-39

```
SQL> SELECT ename, hiredate, LAST_DAY(hiredate), NEXT_DAY(hiredate, '星
期日'),
   2        MONTHS_BETWEEN(SYSDATE, hiredate) "Months",
   3        ADD_MONTHS(hiredate, 3) "Review"
   4  FROM emp;
```

例 4-39 结果

```
ENAME      HIREDATE   LAST_DAY(H NEXT_DAY(H     Months Review
---------- ---------- ---------- ---------- ---------- ------------
SMITH      17-12 月-80 31-12 月-80 21-12 月-80 268.212076 17-3 月 -81
ALLEN      20-2 月 -81 28-2 月 -81 22-2 月 -81 266.115302 20-5 月 -81
WARD       22-2 月 -81 28-2 月 -81 01-3 月 -81 266.050786 22-5 月 -81
JONES      02-4 月 -81 30-4 月 -81 05-4 月 -81 264.695947 02-7 月 -81
MARTIN     28-9 月 -81 30-9 月 -81 04-10 月-81 258.857238 28-12 月-81
BLAKE      01-5 月 -81 31-5 月 -81 03-5 月 -81 263.728205 01-8 月 -81
CLARK      09-6 月 -81 30-6 月 -81 14-6 月 -81 262.470141 09-9 月 -81
SCOTT      19-4 月 -87 30-4 月 -87 26-4 月 -87  192.14756 19-7 月 -87
KING       17-11 月-81 30-11 月-81 22-11 月-81 257.212076 17-2 月 -82
TURNER     08-9 月 -81 30-9 月 -81 13-9 月 -81 259.502399 08-12 月-81
ADAMS      23-5 月 -87 31-5 月 -87 24-5 月 -87        191 23-8 月 -87
JAMES      03-12 月-81 31-12 月-81 06-12 月-81 256.663689 03-3 月 -82
FORD       03-12 月-81 31-12 月-81 06-12 月-81 256.663689 03-3 月 -82
MILLER     23-1 月 -82 31-1 月 -82 24-1 月 -82        255 23-4 月 -82
已选择 14 行。
```

4.8 ROUND和TRUNC函数用于日期型数据

除了可以把 ROUND 和 TRUNC 函数用于数字型数据外，还可以将其用于日期型数据。可以通过例 4-40 的查询语句验证把 ROUND 函数用于日期型数据。

例 4-40

```
SQL> SELECT ROUND('28-OCT-01','MONTH')
2  FROM dual;
```

例 4-40 结果

```
SELECT ROUND('28-OCT-01','MONTH')
```

```
                 *
ERROR 位于第 1 行:
ORA-01722: ????
```

例 4-40 的查询语句之所以产生错误，是因为 28-OCT-01 是字符串，而 ROUND 函数是不能用于字符型数据的，所以要先使用 TO_DATE 函数把字符串转换成日期之后才能用 ROUND 和 TRUNC 函数。

可以通过例 4-41 的含有 TO_DATE 函数的查询语句来验证把 ROUND 函数用于日期型数据。

例 4-41

```
SQL> SELECT ROUND(TO_DATE('28-OCT-01'),'MONTH')
  2  FROM dual;
```

例 4-41 结果

```
ROUND(TO_
---------
01-NOV-01
```

我们在例 4-41 中指明操作的单位为月（MONTH），因为 10 月 28 日为 10 月的下半月，所以 ROUND 函数按四舍五入的规则将 10 月 28 日进为 11 月 1 日，因此，例 3-41 显示的结果表示为：2001 年 11 月 1 日。

也可以指明操作的单位为年。可以通过例 4-42 的含有 TO_DATE 函数的查询语句来验证把 ROUND 函数用于日期型数据。

例 4-42

```
SQL> SELECT ROUND(TO_DATE('28-OCT-01'),'YEAR')
  2  FROM dual;
```

例 4-42 结果

```
ROUND(TO_
---------
01-JAN-02
```

我们在例 4-42 中指明操作的单位为年（YEAR），因为 10 月 28 日为下半年，所以 ROUND 函数按四舍五入的规则将 2001 年 10 月 28 日进为 2002 年 1 月 1 日，因此，例 4-42 显示的结果表示为：2002 年 1 月 1 日。

也可以通过例 4-43 的含有 TO_DATE 函数的查询语句来验证把 TRUNC 函数用于日期型数据。

例 4-43

```
SQL> SELECT TRUNC(TO_DATE('28-OCT-01'),'MONTH')
  2  FROM dual;
```

例 4-43 结果

```
TRUNC(TO_
---------
01-OCT-01
```

我们在例 4-43 中指明操作的单位为月（MONTH），因为 10 月 28 日为 10 月，所以 TRUNC 函数按舍去（截断）的规则将 10 月 28 日截断为 10 月 1 日，因此，例 4-43 显示的结果表示为：2001 年 10 月 1 日。

也可以指明操作的单位为年。您可以通过例 4-44 的含有 TO_DATE 函数的查询语句来验证把 TRUNC 函数用于日期型数据。

例 4-44

```
SQL> SELECT TRUNC(TO_DATE('28-OCT-01'),'YEAR')
  2  FROM dual;
```

例 4-44 结果

```
TRUNC(TO_
---------
01-JAN-01
```

在例 4-44 中指明操作的单位为年（YEAR），因为 2001 年 10 月 28 日为 2001 年，所以 TRUNC 函数按舍去（截断）的规则将 2001 年 10 月 28 日截断为 2001 年 1 月 1 日，因此，例 4-44 显示的结果表示为：2001 年 1 月 1 日。

下面再给出一个综合的例子，如例 4-45 所示。

例 4-45

```
SQL> SELECT ename, hiredate, ROUND(hiredate,'YEAR'), TRUNC(hiredate,'YEAR'
  2        ROUND(hiredate, 'MONTH'), TRUNC(hiredate, 'MONTH')
  3  FROM emp
  4  WHERE hiredate LIKE '%81';
```

例 4-45 结果

```
ENAME                HIREDATE  ROUND(HIR TRUNC(HIR ROUND(HIR TRUNC(HIR
-------------------- --------- --------- --------- --------- ---------
ALLEN                20-FEB-81 01-JAN-81 01-JAN-81 01-MAR-81 01-FEB-81
WARD                 22-FEB-81 01-JAN-81 01-JAN-81 01-MAR-81 01-FEB-81
JONES                02-APR-81 01-JAN-81 01-JAN-81 01-APR-81 01-APR-81
MARTIN               28-SEP-81 01-JAN-82 01-JAN-81 01-OCT-81 01-SEP-81
BLAKE                01-MAY-81 01-JAN-81 01-JAN-81 01-MAY-81 01-MAY-81
CLARK                09-JUN-81 01-JAN-81 01-JAN-81 01-JUN-81 01-JUN-81
KING                 17-NOV-81 01-JAN-82 01-JAN-81 01-DEC-81 01-NOV-81
TURNER               08-SEP-81 01-JAN-82 01-JAN-81 01-SEP-81 01-SEP-81
JAMES                03-DEC-81 01-JAN-82 01-JAN-81 01-DEC-81 01-DEC-81
FORD                 03-DEC-81 01-JAN-82 01-JAN-81 01-DEC-81 01-DEC-81
已选择 10 行。
```

4.9　不同数据类型之间的隐含转换

赋值语句中数据类型之间的隐含转换（Oracle 自动转换）包括：

- 将变长字符型（VARCHAR2）或定长字符型（CHAR）转换成数字型（NUMBER）。
- 将变长字符型（VARCHAR2）或定长字符型（CHAR）转换成日期型（DATE）。
- 将数字型（NUMBER）转换成变长字符型（VARCHAR2）。
- 将日期型（DATE）转换成变长字符型（VARCHAR2）。

表达式中（Oracle 自动转换）的隐含转换包括：

- 将变长字符型（VARCHAR2）或定长字符型（CHAR）转换成数字型（NUMBER）。
- 将变长字符型（VARCHAR2）或定长字符型（CHAR）转换成日期型（DATE）。

在将字符类型的数据转换为数字型时，要保证字符型数据为有效的数。在将字符类型的数据转换为日期型时，要保证字符型数据为有效的日期，否则转换不能成功。

数字型（NUMBER）的数据与日期型（DATE）的数据之间不能进行直接转换，必须将其中的一种数据类型先转换成字符型，之后再将字符型转换成其中的另一种数据类型。

注意：

尽管 Oracle 提供了数据类型之间的隐含转换方法，但读者应该尽量避免使用这种方法。因为用隐含转换方法写出来的 SQL 语句其他人很难理解，随着时间的流逝连您自己都很难理解，特别是当 SQL 语句嵌在大型程序中时。另外，Oracle 推出新的版本时可能会修改一些隐含转换的规则，这会使您的程序移植遇到麻烦。

4.10　不同数据类型之间的显式转换

Oracle 提供了 3 个转换函数来完成不同数据类型之间的显式转换，这 3 个显式转换函数为 TO_CHAR、TO_NUMBER 和 TO_DATE。

下面分别介绍这 3 个转换函数。

1. TO_CHAR(日期,'fmt')

该函数的这种格式把日期型数据转换成变长字符串，其中，fmt 为日期模式。可以通过例 4-46 的查询语句来验证 TO_CHAR 函数如何将日期型数据转换成字符串。

例 4-46

```
SQL> SELECT ename, TO_CHAR(hiredate, 'DD/MM/YY')
  2  FROM emp
  3  WHERE hiredate LIKE '%82';
```

例 4-46 结果

```
ENAME                TO_CHAR(HIREDATE
```

```
-------------------- ----------------
MILLER               23/01/82
```

其中例 4-46 中的“'DD/MM/YY'”为日期模式，它们必须用单引号括起来。

常用的日期模式如下。

- YYYY：完整的年份数字表示（如 2001）。
- YEAR：年份的英文表示（如 NINETEEN EIGHTY-SEVEN）。
- MM：用两位数字来表示月份。
- MONTH：月份完整的英文表示。
- DY：用 3 个英文字符的缩写来表示星期几。
- DAY：星期几的完整的英文表示。
- DD：几号的数字表示。

可以在日期中加入字符串，但必须用双引号括起来。您还可以使用 SP 和 TH，它们分别为数字的英文表示和数字的序数表示。

可以从下面的例子中体会到一些日期模式的具体用法。

可以通过例 4-47 的查询语句产生这样的显示输出：首先以阿拉伯数字显示日（6），之后空一格并以完整的英语显示月（MAY），之后再空一格并在最后以完整的英语显示年（TWO THOUSAND TWO）。

例 4-47

```
SQL> SELECT TO_CHAR(SYSDATE,'fmDD MONTH YEAR')
  2  FROM dual;
```

例 4-47 结果

```
TO_CHAR(SYSDATE,'FMDDMONTHYEAR')
--------------------------------
6 MAY TWO THOUSAND TWO
```

这里的 fm 用来压缩前导 0 或空格。

下面给出一个比较实际的例子。为了能在一行中显示日期数据 HIREDATE，可以使用例 4-48 的 SQL*Plus 命令将 HIREDATE 的输出宽度加长为 60 个字符。

例 4-48

```
SQL> col hiredate for a60
```

然后可以使用例 4-49 的查询语句以得到员工的名字（Name）、工资（Salary）和雇用日期（HIREDATE）的信息。

例 4-49

```
SQL> SELECT  ename "Name", sal "Salary",
  2    TO_CHAR(hiredate, 'fmDdspth "of" Month Year fmHH:MI:SS AM') HIREDATE
  3  FROM emp;
```

例 4-49 结果

```
Name         Salary HIREDATE
-------- ---------- ----------------------------------------------------
SMITH          800  Seventeenth of December Nineteen Eighty 12:00:00 AM
ALLEN         1600  Twentieth of February Nineteen Eighty-One 12:00:00 AM
WARD          1250 Twenty-Second of February Nineteen Eighty-One 12:00:00 AM
JONES         2975  Second of April Nineteen Eighty-One 12:00:00 AM
MARTIN        1250 Twenty-Eighth of September Nineteen Eighty-One 12:00:00 AM
BLAKE         2850  First of May Nineteen Eighty-One 12:00:00 AM
CLARK         2450  Ninth of June Nineteen Eighty-One 12:00:00 AM
SCOTT         3000  Nineteenth of April Nineteen Eighty-Seven 12:00:00 AM
KING          5000  Seventeenth of November Nineteen Eighty-One 12:00:00 AM
TURNER         1500  Eighth of September Nineteen Eighty-One 12:00:00 AM
ADAMS          1100  Twenty-Third of May Nineteen Eighty-Seven 12:00:00 AM
JAMES           950  Third of December Nineteen Eighty-One 12:00:00 AM
FORD           3000  Third of December Nineteen Eighty-One 12:00:00 AM
MILLER         1300  Twenty-Third of January Nineteen Eighty-Two 12:00:00 AM
已选择 14 行。
```

2．TO_CHAR(数字, 'fmt')

该函数的这种格式把数字型数据转换成变长字符串。其中，fmt 为数字模式。常用的数字模式如下。

- 9：一位数字。
- 0：显示前导零。
- $：显示美元号。
- L：显示本地货币号。
- .：显示小数点。
- ，：显示千位符。
- MI：在数的右边显示减号。
- PR：把负数用尖括号括起来。

例 4-50 是一个比较实际的例子。该查询语句用来显示员工的名字（Name）和年薪（Annual Salary），其中显示年薪时要包括美元（$）符、千位符（，）和小数点（.），而且要显示小数点后两位数，即使小数点后的数为 0 也要显示。相信这个例子能帮助您理解 TO_CHAR()函数和数字模式的用法。

例 4-50

```
SQL> SELECT ename "Name", TO_CHAR(sal*12, '$99,999.00') "Annual Salary"
  2  FROM emp;
```

例 4-50 结果

```
Name        Annual Salary
```

```
---------- -------------
SMITH          $9,600.00
ALLEN         $19,200.00
WARD          $15,000.00
JONES         $35,700.00
MARTIN        $15,000.00
BLAKE         $34,200.00
CLARK         $29,400.00
SCOTT         $36,000.00
KING          $60,000.00
TURNER        $18,000.00
ADAMS         $13,200.00
JAMES         $11,400.00
FORD          $36,000.00
MILLER        $15,600.00
已选择 14 行。
```

注意：

当您使用 L 时，在 Oracle 9i 或之前版本中显示的为 RMB，而从 Oracle 10g 开始则为￥。您可以使用例 4-51 的查询语句来验证这一点（以下是 Oracle 9i 或之前版本的显示结果，在 Oracle 10g 中显示结果除将 RMB 换成￥之外，其他完全相同。为了节省篇幅，这里就不给出显示结果了）。

例 4-51

```
SQL> SELECT ename "Name", TO_CHAR(sal*12, 'L99,999.00') "Annual Salary"
  2  FROM emp;
```

例 4-51 结果

```
Name       Annual Salary
---------- --------------------
SMITH                RMB9,600.00
ALLEN               RMB19,200.00
WARD                RMB15,000.00
JONES               RMB35,700.00
MARTIN              RMB15,000.00
BLAKE               RMB34,200.00
CLARK               RMB29,400.00
SCOTT               RMB36,000.00
KING                RMB60,000.00
TURNER              RMB18,000.00
ADAMS               RMB13,200.00
JAMES               RMB11,400.00
```

```
FORD                RMB36,000.00
MILLER              RMB15,600.00
已选择 14 行。
```

如果实际数据的位数超过了您在格式语句中所提供的位数，Oracle Server 又如何处理呢？可以通过例 4-52 的查询语句来看到 Oracle Server 的处理方式（以下是 Oracle 9i 或之前版本的显示结果，在 Oracle 10g 中显示结果除了将 RMB 换成￥之外，其他完全相同。为了节省篇幅，这里也不给出显示结果了）。

例 4-52

```
SQL> SELECT ename "Name", TO_CHAR(sal*12, 'L9,999.00') "Annual Salary"
2  FROM emp;
```

例 4-52 结果

```
Name          Annual Salary
---------- -------------------
SMITH              RMB9,600.00
ALLEN      ###################
WARD       ###################
JONES      ###################
MARTIN     ###################
BLAKE      ###################
CLARK      ###################
SCOTT      ###################
KING       ###################
TURNER     ###################
ADAMS      ###################
JAMES      ###################
FORD       ###################
MILLER     ###################
已选择 14 行。
```

从上面的例子您可以看到，当在把数字转换成字符串显示时其位数一定要留够，否则您只能得到一些“#”号。介绍完了 TO_CHAR()函数，下面我们来介绍 TO_NUMBER 和 TO_DATE 函数。

3．TO_NUMBER(字符串 [,'fmt]')

该函数用于把字符串转换成数字。

4．TO_DATE(字符串 [,'fmt]')

该函数用于把字符串转换成日期型数据。

您在例 4-41~例 4-44 已看到了许多使用 TO_DATE 函数的例子，因此我们就不重复讨论这个函数了。TO_NUMBER 函数比较简单，您可以自己构造几个例子来练习。

您还记得“2000 年问题”吗？由于早期计算机硬件非常昂贵，程序员们在编程时为了节省内存资源就用两位数来表示年份，如 73 表示 1973。随着新世纪的逼近，许多计算机专家们开始意识到问题的严重性。以这种方法开发的软件将无法在本世纪正常地工作，而且可能会造成程序的逻辑混乱，因为这时计算机很可能把 73 识别成 2073 年。这就是著名的“2000 年问题”（虽然“2000 年问题”在国外曾引起很大的恐慌，但实际上几乎没什么事情发生。也许利用人们的恐惧来赚钱才是“2000 年问题”背后的真正原因）。Oracle 引入 RR 日期格式可能是为了解决“2000 年问题”而提出的。

RR 日期格式的算法和实例如下：

（1）如果当前年份的最后两位数（即不包括世纪）为 0~49，并且指定的年份的最后两位数也为 0~49，则返回的日期在本世纪。例如，当前年份为 2002 年，指明的日期是 01-OCT-08，RR 日期格式返回的日期为 2008 年 10 月 1 日，而 YY 日期格式返回的日期也为 2008 年 10 月 1 日。

（2）如果当前年份的最后两位数（即不包括世纪）为 0~49，并且指定的年份的最后两位数为 50~99，则返回的日期为上一世纪。例如，当前年份为 2002 年，指明的日期是 01-OCT-98，RR 日期格式返回的日期为 1998 年 10 月 1 日，而 YY 日期格式返回的日期则为 2098 年 10 月 1 日。这也许就是我们所说的“2000 年问题”。

（3）如果当前年份的最后两位数（即不包括世纪）为 50~99，并且指定的年份的最后两位数为 0~49，则返回的日期为下一世纪。例如，当前年份为 1999 年，指明的日期是 01-OCT-08，RR 日期格式返回的日期为 2008 年 10 月 1 日，而 YY 日期格式返回的日期则为 1908 年 10 月 1 日。

（4）如果当前年份的最后两位数（即不包括世纪）为 50~99，并且指定的年份的最后两位数也为 50~99，则返回的日期在本世纪。例如，当前年份为 1999 年，指明的日期是 01-OCT-98，RR 日期格式返回的日期为 1998 年 10 月 1 日，而 YY 日期格式返回的日期也为 1998 年 10 月 1 日。

下面我们来看一个比较实际的例子。例 4-53 的查询语句是要显示在 1981 年所雇用的所有员工的名字（Name）、工资（Salary）和雇用日期（hiredate）的信息。

例 4-53

```
SQL> SELECT ename "Name", job, sal AS "Salary", hiredate
   2  FROM emp
   3  WHERE hiredate BETWEEN '01-Jan-81' AND '31-Dec-81'
   4  ORDER BY hiredate;
```

例 4-53 结果

```
Name                 JOB                  Salary HIREDATE
-------------------- ------------------ ---------- ---------
ALLEN                SALESMAN               1600 20-FEB-81
WARD                 SALESMAN               1250 22-FEB-81
JONES                MANAGER                2975 02-APR-81
BLAKE                MANAGER                2850 01-MAY-81
```

```
CLARK                MANAGER                 2450  09-JUN-81
TURNER                SALESMAN                1500  08-SEP-81
MARTIN                SALESMAN                1250  28-SEP-81
KING                 PRESIDENT               5000  17-NOV-81
JAMES                CLERK                    950  03-DEC-81
FORD                 ANALYST                 3000  03-DEC-81
已选择 10 行。
```

如果您的数据库为中文，可以将例 4-53 的查询语句修改成例 4-54 的查询语句。

例 4-54

```
SQL> SELECT ename "Name", job, sal AS "Salary", hiredate
  2  FROM emp
  3  WHERE hiredate BETWEEN '01-1 月-81' AND '31-12 月-81'
  4  ORDER BY hiredate;
```

例 4-54 结果

```
Name       JOB          Salary HIREDATE
---------- --------- ---------- -----------
ALLEN      SALESMAN       1600 20-2 月 -81
WARD       SALESMAN       1250 22-2 月 -81
JONES      MANAGER        2975 02-4 月 -81
BLAKE      MANAGER        2850 01-5 月 -81
CLARK      MANAGER        2450 09-6 月 -81
TURNER     SALESMAN       1500 08-9 月 -81
MARTIN     SALESMAN       1250 28-9 月 -81
KING       PRESIDENT      5000 17-11 月-81
JAMES      CLERK           950 03-12 月-81
FORD       ANALYST        3000 03-12 月-81
已选择 10 行。
```

上面的例子在 Oracle 9i 以前的版本上运行可能会得不到任何结果，因为 Oracle 9i 以前的版本默认日期输入/输出格式为 YY，而 emp 表中根本就没有任何 2081 年的记录。从 Oracle 9i 开始，默认日期输入/输出格式已经改为 RR。例 4-55 的查询可能会得不到任何结果，这是因为您将日期格式改为 YY 方式，即 Oracle 9i 以前的版本的默认日期格式。

例 4-55

```
SQL> SELECT ename "Name", job, sal AS "Salary", hiredate
  2  FROM emp
  3  WHERE hiredate BETWEEN TO_DATE('01-Jan-81', 'DD-MON-YY')
  4                     AND TO_DATE('31-Dec-81', 'DD-MON-YY')
  5  ORDER BY hiredate;
```

例 4-55 结果

```
未选定行
```

可以在您的查询语句中显式地指明日期格式为 RR 方式，如例 4-56。

例 4-56

```
SQL> SELECT ename "Name", job, sal AS "Salary", hiredate
  2  FROM emp
  3  WHERE hiredate BETWEEN TO_DATE('01-Jan-81', 'DD-MON-RR')
  4                 AND TO_DATE('31-Dec-81', 'DD-MON-RR')
  5  ORDER BY hiredate;
```

例 4-56 结果

```
Name                 JOB                  Salary HIREDATE
-------------------- ------------------ ---------- ---------
ALLEN                SALESMAN              1600  20-FEB-81
WARD                 SALESMAN              1250  22-FEB-81
JONES                MANAGER               2975  02-APR-81
BLAKE                MANAGER               2850  01-MAY-81
CLARK                MANAGER               2450  09-JUN-81
TURNER               SALESMAN              1500  08-SEP-81
MARTIN               SALESMAN              1250  28-SEP-81
KING                 PRESIDENT             5000  17-NOV-81
JAMES                CLERK                  950  03-DEC-81
FORD                 ANALYST               3000  03-DEC-81
已选择 10 行。
```

该例子的显示结果与例 4-53 完全相同，这是因为 Oracle 9i 的默认日期输入/输出格式为 RR。

注意：

请读者尽量避免使用两位数来表示年份，即尽量把年份写全了（要用 2002 不要用 02，要用 1998 不要用 98）。不要费了半天劲，省的钱还不够打瓶醋呢！我们生活在新旧世纪交替之际，不是我们高瞻远瞩，而是不想自找麻烦。

到此为止，您已经学习了全部常用的单行函数，熟练掌握这些函数的最好最快的方法就是实践。

建议：

读者将本书中的多数例子，最好是全部的例子在计算机上试一下，这样您的体会就会更深一些。这就像学习游泳一样，无论您遇到多么好的教练，看过多少学习游泳的录像，如果不跳到水中（当然不包括浴缸），永远也学不会。数据库是一门实践性很强的课程。如果您没有在计算机上操作过数据库，就很难真正地学好这门课。

4.11　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 什么是函数？
- 单行函数。
- 可以用单行函数的子句有哪些？
- 常用的字符型函数。
- 常用的数字型函数。
- 常用的日期型函数。
- 3 个转换型函数。
- Oracle 系统的虚表 dual。
- 什么是 SQL 语言？
- 什么是 SQL*Plus？
- 日期型数据。
- TO_CHAR 函数中的日期模式。
- 如何将 ROUND 函数用于日期型数据？
- 如何将 TRUNC 函数用于日期型数据？
- TO_CHAR 函数中的数字模式。
- 数据类型之间的隐含转换。
- 数据类型之间的显式转换。
- 日期的 RR 格式。
- 日期的 YY 格式。

第5章

NULL 值的处理、逻辑操作和函数嵌套

在不少有关 Oracle SQL 的书中，NULL 值的处理、逻辑操作和函数嵌套这 3 个内容是放在不同的章中介绍的。但我本人认为它们的联系十分紧密，应该放在一起讨论，希望这样的安排能够帮助读者理解这些比较难的内容。

5.1 什么是空值

在介绍空值（NULL）之前，我们先看例 5-1。

例 5-1

```
SQL> SELECT ename, job, sal, comm.
  2  FROM emp
  3  WHERE job IN ('CLERK','SALESMAN')
  4  ORDER BY job;
```

例 5-1 结果

```
ENAME      JOB              SAL       COMM
---------- ---------- ---------- ----------
SMITH      CLERK            800
ADAMS      CLERK           1100
MILLER     CLERK           1300
JAMES      CLERK            950
ALLEN      SALESMAN        1600        300
WARD       SALESMAN        1250        500
MARTIN     SALESMAN        1250       1400
TURNER     SALESMAN        1500          0
已选择 8 行。
```

请看例 5-1 查询结果中的 COMM（佣金）这一列，所有的 CLERK（文员）的 COMM 这一列都没有任何显示。但 SALESMAN 却都有值（有可能是 0）。这是因为 CLERK 根本没有佣金，即这一列根本就不适用于 CLERK。Oracle 系统的设计者把它们置为 NULL。

NULL 值是一个很特别的值。它既不是 0，也不是空格。它的值是没有定义的、未知的、不确定的。一些英文书中用了如下的词来形容 NULL“unavailable，unassigned，undefined，unknown，immeasurable，inapplicable”。总之您没有办法得到它的准确值。

现在用一个世俗的例子来说明空值 NULL。我们常常在电视剧中看到一位英俊的少年向一位妙龄女郎求爱的镜头。他可能会说“我将把我全部的爱都奉献给你”。现在您能告诉我他的爱的市场价值吗?

我相信没人能说出他的爱到底值多少钱，也可能是一钱不值的谎言，也可能是价值无限的，即当她遇到危难时他会挺身而出，甚至用他的血肉之躯为她挡上几十刀或几百枪。

但是如果他说：“为了表达我对你的崇高而珍贵的爱，我现在把这辆宝马（BMW）献给你。”现在您可以很容易地得到这份珍贵的爱的市场价值。现在您能理解 NULL 的含义了吗?

造成这种现象的主要原因是信息不完全。设想一下一个罪犯追踪系统。我们知道一个人的性别不是男就是女，不可能是不男不女。但当一件大案发生时，警方可能对是谁做的案一无所知，当然也就无法决定罪犯的性别了。此时不是罪犯为不男不女，而是警方不知道罪犯是男还是女，所以只能把相应表中的性别一栏置为 NULL。除了性别外，有关此案可能还有许多疑团。随着侦探工作的进展，不少疑团被解开，警方已能断定罪犯的性别，即从未知变成了已知，也就是由 NULL 变成了男或女。

5.2　含有空值的表达式的运算

通过以上的讨论，您应该已经知道了 NULL 值的引入有它现实的基础，并且也为数据库系统的设计提供了方便，但它也带来了一些麻烦。

假设您的老板让您打印一份不包括推销员的员工收入（工资+奖金）清单，他想知道他在这方面的支出，看看能否在这方面节省一些开支以渡过眼下的困境。于是您给出了例 5-2 的查询语句。

例 5-2

```
SQL> SELECT ename "Name", sal+comm "Income", job
   2  FROM emp
   3  WHERE job NOT LIKE 'SALES%'
  4  ORDER BY job;
```

例 5-2 结果

```
Name                     Income  JOB
-------------------- ---------- ---------
SCOTT                            ANALYST
FORD                             ANALYST
SMITH                            CLERK
JAMES                            CLERK
ADAMS                            CLERK
MILLER                           CLERK
JONES                            MANAGER
BLAKE                            MANAGER
CLARK                            MANAGER
```

```
KING                          PRESIDENT
已选择 10 行。
```

例 5-2 所得的结果令您大吃一惊。居然所有员工的收入都为 0（除了推销员之外），这真是太不可思议了，因为在经济如此发达的 21 世纪，您的老板竟能雇用如此之多的义工，这可能吗？

问题出在 NULL（空值）上，因为只要在一个表达式中包含任何 NULL（空值），该表达式的值就为 NULL。我们回到本章开始时的那个世俗的故事。经过了一段时间，那位美貌的女郎做了一件令那位少年非常伤心的事，那位少年可能气愤地对她说："我对你的爱已减了一成（10%）。"您能告诉我他的爱的市场价值吗？除了天知、地知和他知（没准他自己也不知），没人能知，即答案还是 NULL（空值）。

NULL（空值）的独到之处还不仅如此。您的老板现在让您打印一份不挣佣金的员工名单。于是您输入了例 5-3 的查询语句。

例 5-3

```
SQL> SELECT empno, ename, sal, job, comm
   2  FROM emp
   3  WHERE comm = NULL;
```

例 5-3 结果

```
未选定行。
```

结果又使您感到非常意外，因为 Oracle 没找到任何不挣佣金的员工。您仔细地检查了以上查询语句但没有发现任何语法错误。于是您试着用例 5-4 的查询语句打印一份挣佣金的员工名单。

例 5-4

```
SQL> SELECT empno, ename, sal, job, comm
   2  FROM emp
   3  WHERE comm != NULL;
```

例 5-4 结果

```
未选定行。
```

其结果同样让您大失所望。这是因为您无法证明 NULL 等于某个值，也无法证明 NULL 不等于某个值。我们又回到本章开始时的那个世俗的故事。假设又有一位白马王子向那位美丽动人的少女说了同样的爱的誓言。您能告诉我他们的誓言是相等还是不等呢？我相信没人能回答这个问题。

为了解决这个进退两难的问题，Oracle 引入了一个新的运算符 IS NULL 来处理这类问题。现在您可以使用 IS NULL 重写例 5-3 的查询语句，如例 5-5 所示。

例 5-5

```
SQL> SELECT empno, ename, sal, job, comm
   2  FROM emp
   3  WHERE comm IS NULL;
```

例 5-5 结果

```
     EMPNO ENAME                                 SAL JOB                          COMM
--------- ---------------- ----------- ------------------ -----------
      7369 SMITH                                 800 CLERK
      7566 JONES                                2975 MANAGER
      7698 BLAKE                                2850 MANAGER
      7782 CLARK                                2450 MANAGER
      7788 SCOTT                                3000 ANALYST
      7839 KING                                 5000 PRESIDENT
      7876 ADAMS                                1100 CLERK
      7900 JAMES                                 950 CLERK
      7902 FORD                                 3000 ANALYST
      7934 MILLER                               1300 CLERK
已选择 10 行。
```

这次得到了您所希望的结果。现在可以用 IS NOT NULL 重写例 5-4 的查询语句，如例 5-6 所示。

例 5-6

```
SQL> SELECT empno, ename, sal, job, comm
  2  FROM emp
  3  WHERE comm IS NOT NULL;
```

例 5-6 结果

```
     EMPNO ENAME                                 SAL JOB                          COMM
---------- -------------------- ---------- -------------- -------
      7499 ALLEN                                1600 SALESMAN                      300
      7521 WARD                                 1250 SALESMAN                      500
      7654 MARTIN                               1250 SALESMAN                     1400
      7844 TURNER                               1500 SALESMAN                        0
```

例 5-6 的显示结果给出了所有挣佣金的员工的名单。

5.3　空值的排序

现在通过例子来介绍 NULL 值的排序。您可以输入例 5-7 所示的带有 ORDER BY 子句的查询语句。

例 5-7

```
SQL> SELECT ename, job, comm
  2  FROM emp
  3  ORDER BY comm;
```

例 5-7 结果

```
ENAME                JOB                COMM
-------------------- ------------------ ----------
TURNER                SALESMAN                 0
ALLEN                SALESMAN                300
WARD                SALESMAN                 500
MARTIN                SALESMAN              1400
SMITH                CLERK
JONES                MANAGER
JAMES                CLERK
MILLER                CLERK
FORD                ANALYST
ADAMS                CLERK
BLAKE                MANAGER
CLARK                MANAGER
SCOTT                ANALYST
KING                PRESIDENT
已选择 14 行。
```

例 5-7 的显示结果表明，在升序排序时，NULL 值排在最后。

为了进一步解释有关 NULL 值的排序的问题。您可以再输入例 5-8 所示的带有 ORDER BY 子句和 DESC 关键字的查询语句。

例 5-8

```
SQL> SELECT ename, job, comm
  2  FROM emp
  3  ORDER BY comm DESC;
```

例 5-8 结果

```
ENAME                JOB              COMM
-------------------- --------------- ---------
SMITH                CLERK
JONES                MANAGER
CLARK                MANAGER
BLAKE                MANAGER
SCOTT                ANALYST
KING                PRESIDENT
JAMES                CLERK
MILLER                CLERK
FORD                ANALYST
ADAMS                CLERK
MARTIN                SALESMAN            1400
WARD                SALESMAN              500
```

```
ALLEN                 SALESMAN              300
TURNER                SALESMAN                0
已选择 14 行。
```

例 5-7 显示的结果表明，在降序排序时，NULL 值排在最前。

5.4　逻辑表达式和逻辑运算符

Oracle 提供了 AND（逻辑与/逻辑乘）、OR（逻辑加/逻辑或）和 NOT（逻辑非）3 个逻辑运算符。其中，AND（逻辑与/逻辑乘）和 OR（逻辑加/逻辑或）用于把两个条件组合在一起产生一个结果。

其格式如下：

条件 1　逻辑运算符　条件 2;

它也叫逻辑表达式，这里逻辑运算符为 AND 或 OR。

逻辑表达式有以下的重要特性：

条件 1　逻辑运算符　条件 2 =条件 2　逻辑运算符　条件 1　　（定理 5.1）

在二值逻辑中逻辑表达式或条件只能为真（T）或假（F），但在 Oracle 的逻辑表达式中还引入了另一个值——未知（NULL）。

在二值逻辑中逻辑表达式“条件 1　AND　条件 2”中，只有当条件 1 和条件 2 同时为真时其结果才为真，否则为假。但现在我们多了一个 NULL 值，以上逻辑表达式的结果又该如何呢？

下面是 AND 的真值表（算法）：

F AND F = F	F AND T = F	F AND NULL = F
T AND F = F	**T AND T = T**	T AND NULL IS NULL
NULL AND F = F	**NULL AND T IS NULL**	**NULL AND NULL IS NULL**

您只要能记住真值表的中线和左下角（即黑体）部分就可以了。因为剩余部分可以用定理 5.1 推导出来。您也可以把下划线部分看成对称轴，对称轴上、下两部分的结果相等。因此，您只要记住对称轴的上部分或下部分就可以了，另一部分您可以用交换 AND 左、右的真值来得到。

AND 运算的优先级为：

F ⟶ NULL ⟶ T

即在 AND 逻辑表达式中，只要有 F 其结果就为 F，如真值表中第 1 行和第 1 列所表示的；如果没有 F，在 AND 逻辑表达式中有 NULL 其结果就为 NULL，如真值表中，除了结果为 F 的部分最后一行和最后一列所表示的；只有当两个条件都为 T 时，AND 逻辑表达式的结果才为 T，如真值表中正中心所表示的。

我们再举一个世俗的例子。设想一位妙龄女郎想用她的青春赌未来，她在某报纸上登了一份征婚广告，要求应征者必须至少有一千万存款并且有发达国家的 PR（永久居留权）。这就是一个逻辑与运算，只有当应征者具备了以上两个条件时才有可能成为这位绝代佳人

的夫君。如果有一位应征者现在是一个名副其实的千万富翁，但正在申请新西兰的移民，Oracle 如何处理这种情况呢？因为他的移民申请能否被批准是一个未知数，所以其值为 NULL。整个逻辑表达式为：T AND NULL = NULL。因此她只有等到信息完全了之后再做决定。

还记得第 2 章中的例 2-1 和例 2-8 吗？在例 2-1 中您的老板想知道谁的工资在 1500 元以上。在例 2-8 中您的老板想知道他雇了多少推销员、文员和经理。您现在可以试着利用逻辑运算符 AND 把它们合成一个查询语句，如例 5-9 所示。

例 5-9

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE sal >= 1500
  4  AND job IN ('SALESMAN', 'CLERK', 'MANAGER')
  5  ORDER BY job;
```

例 5-9 结果

```
     EMPNO ENAME                       SAL JOB
---------- -------------------- ---------- ---------
      7566 JONES                      2975 MANAGER
      7698 BLAKE                      2850 MANAGER
      7782 CLARK                      2450 MANAGER
      7499 ALLEN                      1600 SALESMAN
      7844 TURNER                     1500 SALESMAN
```

上述查询的结果与老板的要求有些差距。我们在修改这个查询语句之前先介绍一下逻辑运算符 OR。

OR 的真值表如下：

<u>T OR T = T</u>	T OR F = T	T OR NULL = T
F OR T = T	**<u>F OR F = F</u>**	F OR NULL IS NULL
NULL OR T = T	**NULL OR F IS NULL**	**<u>NULL AND NULL IS NULL</u>**

同样您只要能记住真值表的中线和左下角（即黑体）部分就可以了。因为剩余部分可以用定理 5.1 推导出来。您也可以把下划线部分看成对称轴，对称轴上、下两部分的结果相等。因此，您只要记住对称轴的上部分或下部分就可以了，另一部分您可以用交换 OR 左、右的真值来得到。

OR 运算的优先级为：

T ——→ NULL ——→ F

即在 OR 逻辑表达式中，只要有 T 其结果就为 T，如真值表中第 1 行和第 1 列所表示的；如果没有 T，在 OR 逻辑表达式中有 NULL 其结果就为 NULL，如真值表中，除了结果为 T 的部分最后一行和最后一列所表示的；只有当两个条件都为 F，OR 逻辑表达式的结果才为 F，如真值表中正中心所表示的。

下面我们再回到那个世俗的例子。由于市场不景气，那位妙龄女郎的征婚广告登了一

段时间，应征者寥寥无几，能满足两个条件的人又都长得“歪瓜裂枣”，带出去实在对不起观众，她不得不面对现实。她的新一期广告要求应征者必须至少有一千万存款或者有发达国家的 PR（永久居留权）。这已是一个逻辑或运算了。现在上面所谈到的千万富翁就可以幸运地与这位美丽而贤慧的妻子白头偕老了。

在例 5-9 的查询语句中，将 WHERE 子句中的 AND 改成 OR，就可得到老板所需要的结果，如例 5-10 所示。

例 5-10

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE sal >= 1500
  4  OR job IN ('SALESMAN', 'CLERK', 'MANAGER')
  5  ORDER BY job;
```

例 5-10 结果

```
     EMPNO ENAME             SAL JOB
---------- ---------- ---------- ---------
      7788 SCOTT            3000 ANALYST
      7902 FORD             3000 ANALYST
      7369 SMITH             800 CLERK
      7876 ADAMS            1100 CLERK
      7934 MILLER           1300 CLERK
      7900 JAMES             950 CLERK
      7566 JONES            2975 MANAGER
      7782 CLARK            2450 MANAGER
      7698 BLAKE            2850 MANAGER
      7839 KING             5000 PRESIDENT
      7499 ALLEN            1600 SALESMAN
      7654 MARTIN           1250 SALESMAN
      7844 TURNER           1500 SALESMAN
      7521 WARD             1250 SALESMAN
已选择 14 行。
```

我们现在介绍最后一个逻辑运算符 NOT。在二值逻辑中，NOT 的真值表非常简单，其真值表如下：

NOT T = F　　　NOT F = T

但在 Oracle 中，还有空值（NULL）。您能说出 NOT NULL 等于什么吗？

我们回到本章开始时的那个世俗的例子。经过了一段风风雨雨，那位少女做了不少令那位少年心碎的事，他终于忍无可忍地对她说：“我不再爱你了。这个宇宙中我最恨的就是你。”您能告诉我，他的恨有多少吗？我相信没有人能得到它的市场价值。所以 NOT NULL 的结果也为 NULL。NOT 逻辑运算符我们已用过一些了，如 NOT LIKE、IS NOT NULL 及 NOT BETWEEN AND 等。

5.5 运算符的优先级

我们已介绍了许多运算，它们的优先级关系如下：

（1）算术运算符→（2）连接运算符→（3）比较（关系）运算符→（4）IS NULL、IS NOT NULL、LIKE、NOT LIKE、IN、NOT IN 运算符→（5）BETWEEN 和 NOT BETWEEN 运算符→（6）NOT 逻辑运算符→（7）AND 逻辑运算符→（8）OR 逻辑运算符。

如果使用了括号，括号中的运算优先。我们用下面的查询语句（例 5-11）来说明运算优先级。

例 5-11

```
SQL> SELECT empno, ename, sal, job
   2  FROM emp
   3  WHERE job = 'CLERK'
   4  OR    job = 'SALESMAN'
   5  AND   sal >= 1300;
```

例 5-11 结果

```
     EMPNO ENAME                       SAL JOB
---------- -------------------- ---------- --------
      7369 SMITH                       800 CLERK
      7499 ALLEN                      1600 SALESMAN
      7844 TURNER                     1500 SALESMAN
      7876 ADAMS                      1100 CLERK
      7900 JAMES                       950 CLERK
      7934 MILLER                     1300 CLERK
已选择 6 行。
```

按照运算符的优先级，Oracle 首先执行由黑体部分组成的条件，即首先找到工资在 1300 元或以上的推销员（SALESMAN）；第 2 步是找到所有的文员（CLERK）；最后 Oracle 显示的结果为所有的文员（CLERK）和工资在 1300 元或以上的推销员（SALESMAN）的信息。

下面我们用括号改变 WHERE 子句执行的顺序，产生例 5-12 的查询语句。

例 5-12

```
SQL> SELECT empno, ename, sal, job
   2  FROM emp
   3  WHERE (job = 'CLERK'
   4  OR    job = 'SALESMAN')
   5  AND   sal >= 1300;
```

例 5-12 结果

```
     EMPNO ENAME                       SAL JOB
---------- -------------------- ---------- --------
      7499 ALLEN                      1600 SALESMAN
```

```
      7844 TURNER                       1500 SALESMAN
      7934 MILLER                       1300 CLERK
```

按照运算符的优先级，Oracle 首先执行由黑体部分组成的条件，即找出所有的文员（CLERK）和推销员（SALESMAN），接下来找出工资在 1300 元或以上的员工；最后 Oracle 显示的结果为所有工资在 1300 元或以上的文员（CLERK）和推销员（SALESMAN）的信息。

5.6 用AND和OR替代BETWEEN AND和IN运算符

您在第 2 章的例 2-2 中使用了例 5-13 的查询语句。

例 5-13

```
SQL> SELECT empno, ename, sal
  2  FROM emp
  3  WHERE sal BETWEEN 1500 AND 2900;
```

例 5-13 结果

```
     EMPNO ENAME                       SAL
---------- -------------------- ----------
      7499 ALLEN                      1600
      7698 BLAKE                      2850
      7782 CLARK                      2450
      7844 TURNER                     1500
```

您可以使用例 5-14 的查询语句来得到和例 5-13 完全相同的结果。

例 5-14

```
SQL> SELECT empno, ename, sal
  2  FROM emp
  3  WHERE sal >= 1500
  4  AND   sal <= 2900;
```

例 5-14 结果

```
     EMPNO ENAME                       SAL
---------- -------------------- ----------
      7499 ALLEN                      1600
      7698 BLAKE                      2850
      7782 CLARK                      2450
      7844 TURNER                     1500
```

您在第 2 章的例 2-8 中使用了例 5-15 的查询语句。

例 5-15

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE job IN ('SALESMAN', 'CLERK', 'MANAGER');
```

例 5-15 结果

```
     EMPNO ENAME             SAL JOB
---------- ---------- ---------- ---------
      7369 SMITH             800 CLERK
      7499 ALLEN            1600 SALESMAN
      7521 WARD             1250 SALESMAN
      7566 JONES            2975 MANAGER
      7654 MARTIN           1250 SALESMAN
      7698 BLAKE            2850 MANAGER
      7782 CLARK            2450 MANAGER
      7844 TURNER           1500 SALESMAN
      7876 ADAMS            1100 CLERK
      7900 JAMES             950 CLERK
      7934 MILLER           1300 CLERK
已选择 11 行。
```

您可以使用例 5-16 的查询语句来得到和例 5-15 完全相同的结果。

例 5-16

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE job = 'SALESMAN'
  4  OR    job = 'CLERK'
  5  OR    job = 'MANAGER';
```

例 5-16 结果

```
     EMPNO ENAME             SAL JOB
---------- ---------- ---------- ---------
      7369 SMITH             800 CLERK
      7499 ALLEN            1600 SALESMAN
      7521 WARD             1250 SALESMAN
      7566 JONES            2975 MANAGER
      7654 MARTIN           1250 SALESMAN
      7698 BLAKE            2850 MANAGER
      7782 CLARK            2450 MANAGER
      7844 TURNER           1500 SALESMAN
      7876 ADAMS            1100 CLERK
      7900 JAMES             950 CLERK
      7934 MILLER           1300 CLERK
已选择 11 行。
```

例 5-14 和例 5-16 说明，我们可以用 AND 和 OR 替代 BETWEEN AND 和 IN 运算符，只是后者对非计算机人员来说更容易理解而已。通常要得到某一结果，可能有多种查询的

方法，但往往每个查询语句的效率是不同的。

5.7　NVL函数

NVL 函数是一个空值转换函数。您在第 5 章例 5-2 中使用了例 5-17 的查询语句。

例 5-17

```
SQL> SELECT ename "Name", sal+comm "Income", job
    2  FROM emp
    3  WHERE job NOT LIKE 'SALES%'
    4  ORDER BY job;
```

例 5-17 结果

```
Name                      Income JOB
-------------------- ---------- ---------
SCOTT                           ANALYST
FORD                            ANALYST
SMITH                           CLERK
JAMES                           CLERK
ADAMS                           CLERK
MILLER                          CLERK
JONES                           MANAGER
BLAKE                           MANAGER
CLARK                           MANAGER
KING                            PRESIDENT
已选择 10 行。
```

由于表达式 sal+comm 中 comm 的值为 NULL，我们没得到 Income。Oracle 提供的 NVL 函数可以用来解决这一难题。

您可以将例 5-17 的查询语句改写成例 5-18 的 SQL 语句。

例 5-18

```
SQL> SELECT ename "Name", sal+NVL(comm,0) "Income", job
    2  FROM emp
    3  WHERE job NOT LIKE 'SALES%'
    4  ORDER BY job;
```

例 5-18 结果

```
Name            Income JOB
---------- ---------- ---------
SCOTT             3000 ANALYST
FORD              3000 ANALYST
SMITH              800 CLERK
```

```
JAMES              950  CLERK
ADAMS             1100  CLERK
MILLER            1300  CLERK
JONES             2975  MANAGER
BLAKE             2850  MANAGER
CLARK             2450  MANAGER
KING              5000  PRESIDENT
已选择 10 行。
```

从例 5-18 您可能已经看出，NVL 函数用于把一空值（NULL）转换成某一实际的值，它的格式如下：

NVL(表达式 1，表达式 2)

如果表达式 1 为空值（NULL），NVL 返回值为表达式 2 的值，否则返回表达式 1 的值。

该函数的目的是把一个空值（NULL）转换成一个实际的值。其表达式 1 和表达式 2 可以是数字型、字符型或日期型，但表达式 1 和表达式 2 的数据类型必须一致。例如，

- 对数字型：NVL(comm,0)。
- 对字符型：NVL(TO_CHAR(comm), 'No Commission')。
- 对日期型：NVL(hiredate, '31-DEC-99')。

有经验的 Oracle 开发人员会在他们开发的 SQL 语句中经常地使用 NVL 函数，以避免因空值（NULL）产生的错误。

5.8 DECODE函数

因为 SQL 中没有逻辑判断语句（分支语句），所以 Oracle 引入了 DECODE 函数来完成类似的功能。您可以使用例 5-19 的查询语句，利用 DECODE 函数求出基于不同职位（JOB）的每个员工加薪后的工资值。

例 5-19

```
SQL> SELECT ename "Name", job, sal "Salary",
          DECODE(job, 'SALESMAN', sal*1.15,
                      'CLERK',    sal*1.20,
                      'ANALYST',  sal*1.25,
                                  sal*1.40) "New Salary"
     FROM emp
     ORDER BY job;
```

例 5-19 结果

```
Name       JOB           Salary New Salary
---------- --------- ---------- ----------
SCOTT      ANALYST         3000       3750
FORD       ANALYST         3000       3750
SMITH      CLERK            800        960
```

```
ADAMS      CLERK        1100     1320
MILLER     CLERK        1300     1560
JAMES      CLERK         950     1140
JONES      MANAGER      2975     4165
CLARK      MANAGER      2450     3430
BLAKE      MANAGER      2850     3990
KING       PRESIDENT    5000     7000
ALLEN      SALESMAN     1600     1840
MARTIN     SALESMAN     1250   1437.5
TURNER     SALESMAN     1500     1725
WARD       SALESMAN     1250   1437.5
已选择 14 行。
```

在例 5-19 的查询语句中，DECODE 函数执行的方法如下：

（1）当 JOB 为 SALESMAN 时，DECODE 函数返回表达式 sal*1.15 的值，否则，执行步骤（2）。

（2）当 JOB 为 CLERK 时，DECODE 函数返回表达式 sal*1.20 的值，否则，执行步骤（3）。

（3）当 JOB 为 ANALYST 时，DECODE 函数返回表达式 sal*1.25 的值，否则，执行步骤（4）。

（4）DECODE 函数返回表达式 sal*1.40 的值。

5.9　单值函数的嵌套

Oracle 的单值函数可以嵌套。函数的计算次序为从里到外。按 Oracle 的说法，Oracle 的单值函数可以嵌套任意层。但这只是在理论上行得通，因为嵌套的层数要受实际可用内存的限制，而实际可用内存又受操作系统和计算机硬件的限制。

您可以使用例 5-20 的查询语句来测试单值函数的嵌套，该查询语句为一个两层嵌套。

例 5-20

```
SQL> SELECT ename "Name", NVL(TO_CHAR(comm), ename||' is not a Salesperson!')
          "Commission"
  2  FROM emp
  3  ORDER BY 2;
```

例 5-20 结果

```
Name       Commission
---------- ----------------------------
TURNER     0
MARTIN     1400
ALLEN      300
```

```
WARD        500
ADAMS       ADAMS is not a Salesperson!
BLAKE       BLAKE is not a Salesperson!
CLARK       CLARK is not a Salesperson!
FORD        FORD is not a Salesperson!
JAMES       JAMES is not a Salesperson!
JONES       JONES is not a Salesperson!
KING        KING is not a Salesperson!
MILLER      MILLER is not a Salesperson!
SCOTT       SCOTT is not a Salesperson!
SMITH       SMITH is not a Salesperson!
已选择 14 行。
```

在例 5-20 的查询语句中，表达式 NVL(TO_CHAR(comm), ename||' is not a Salesperson!')的计算次序如下：

（1）Oracle 首先用 TO_CHAR 函数把 comm 由数字型转换成字符型。

（2）NVL 函数测试 TO_CHAR(comm)。

（3）如果 TO_CHAR(comm)不为空就返回 TO_CHAR(comm)的值。

（4）如果 TO_CHAR(comm)为空就返回表达式 ename||' is not a Salesperson!'的值。

☠ **注意：**

由于 TO_CHAR 函数把 comm 由数字型转换成字符型，因此，查询显示的结果是按 ASCII 码的顺序排列的。

您也可以在例 5-20 的查询语句中使用别名来排序，修改后的查询语句如例 5-21 所示。

例 5-21

```
SQL> SELECT ename "Name", NVL(TO_CHAR(comm), ename||' is not a Salesperson!')
           "Commission"
  2  FROM emp
  3  ORDER BY "Commission";
```

例 5-21 结果

```
Name        Commission
----------  ---------------------------
TURNER       0
MARTIN       1400
ALLEN        300
WARD         500
ADAMS        ADAMS is not a Salesperson!
BLAKE        BLAKE is not a Salesperson!
CLARK        CLARK is not a Salesperson!
FORD         FORD is not a Salesperson!
```

```
JAMES      JAMES is not a Salesperson!
JONES      JONES is not a Salesperson!
KING       KING is not a Salesperson!
MILLER     MILLER is not a Salesperson!
SCOTT      SCOTT is not a Salesperson!
SMITH      SMITH is not a Salesperson!
已选择 14 行。
```

可以发现例 5-21 显示的结果与例 5-20 完全相同。

如果您想在计算机上练习下一节的例子，您应该在计算机上安装 Oracle 9.0 或以上的版本。

5.10　Oracle 9i新增加的单值函数和表达式

NVL2、NULLIF 和 COALESCE 为 Oracle 9i 新增加的函数，而 CASE 为 Oracle 9i 新增加的表达式。如果读者在阅读以下的介绍时，有看不懂的地方，请不要担心，因为并没有对这些函数和表达式引入新功能，它们的功能都可以用您以前所学的知识完成，只是用它们可能实现起来更容易些。换句话说，即使没有掌握这些函数和表达式，您也可以写出功能完善的 SQL 语句。

建议：

如果读者在阅读这部分内容时感觉到很困难，则可以在没有完全理解这部分内容的情况下继续阅读本书的后续章节。等读者有了一定的编写 SQL 语句的经验之后再来阅读这部分内容就会很容易理解了。

NVL2 是 Oracle 9i 刚引入的一个新函数，它对 NVL 函数进行了小小的扩充。您可以使用例 5-22 的查询语句来测试该函数。

例 5-22

```
SQL> SELECT ename "Name", NVL2(comm,'sal+comm',sal) "Income", job
  2  FROM emp
  3  WHERE job NOT LIKE 'SALES%'
  4  ORDER BY job;
```

例 5-22 结果

```
Name       Income                            JOB
---------- --------------------------------- ---------
SCOTT      3000                              ANALYST
FORD       3000                              ANALYST
SMITH      800                               CLERK
JAMES      950                               CLERK
ADAMS      1100                              CLERK
```

```
MILLER      1300                                     CLERK
JONES       2975                                     MANAGER
BLAKE       2850                                     MANAGER
CLARK       2450                                     MANAGER
KING        5000                                     PRESIDENT
已选择 10 行。
```

在例 5-22 的查询语句中，表达式 NVL2(comm,'sal+comm',sal)"Income" 的执行次序如下：

（1）NVL2 函数测试 comm。

（2）如果 comm 不为空，就返回表达式 sal+comm 的值。

（3）如果 comm 为空，就返回 sal 的值。

（4）返回值的列标为 Income。

NVL2 函数的格式如下：

NVL2(表达式 1，表达式 2，表达式 3)

函数 NVL2(表达式 1，表达式 2，表达式 3)的执行次序如下：

（1）如果表达式 1 不为空值（NULL），NVL2 函数返回值为表达式 2 的值。

（2）如果表达式 1 为空值（NULL），NVL2 函数返回值为表达式 3 的值。

表达式 2 和表达式 3 可以是除 LONG 以外的任何数据类型，表达式 1 可以是任何数据类型。

如果表达式 2 和表达式 3 的数据类型不同，Oracle 服务器把表达式 3 的数据类型转换成表达式 2 的数据类型；此时如果表达式 3 为空值，Oracle 服务器就不进行数据类型的转换。NVL2 函数返回值的数据类型与表达式 2 的数据类型相同。但当表达式 2 的数据类型为定长字符型 CHAR 时，NVL2 函数返回值的数据类型为变长字符型 VARCHAR2。

NULLIF 是 Oracle 9i 新引入的另一个函数，您可以使用例 5-23 的查询语句来测试该函数。

例 5-23

```
SQL> SELECT ename, job,  LENGTH(ename) "Name_Length", LENGTH(job)"Job_Lenght",
  2         NULLIF(LENGTH(ename),LENGTH(job))  "Comparision"
3  FROM emp;
```

例 5-23 结果

```
ENAME      JOB        Name_Length Job_Lenght Comparision
---------- ---------- ----------- ---------- -----------
SMITH      CLERK                5          5
ALLEN      SALESMAN             5          8           5
WARD       SALESMAN             4          8           4
JONES      MANAGER              5          7           5
MARTIN     SALESMAN             6          8           6
BLAKE      MANAGER              5          7           5
CLARK      MANAGER              5          7           5
SCOTT      ANALYST              5          7           5
KING       PRESIDENT            4          9           4
```

```
TURNER     SALESMAN        6          8          6
ADAMS      CLERK           5          5
JAMES      CLERK           5          5
FORD       ANALYST         4          7          4
MILLER     CLERK           6          5          6
已选择 14 行。
```

通过例 5-23 可以看出 NULLIF 函数的功能如下：

- 当 LENGTH(ename)和 LENGTH(job)相等时，函数 NULLIF(LENGTH(ename)和 LENGTH(job))返回空值。
- 否则返回 LENGTH(ename)的值，即 ename 的长度。

NULLIF 函数的格式如下：

NULLIF(表达式 1, 表达式 2)

函数 NULLIF(表达式 1, 表达式 2)的执行次序如下：

（1）NULLIF 函数比较表达式 1 和表达式 2。

（2）如果两个表达式相等就返回空值（NULL）。

（3）如果不等就返回表达式 1。

NULLIF 函数中的表达式 1 不能为 NULL。

现在来介绍 Oracle 9i 引入的另一个新函数 COALESCE。为了使读者容易理解该函数的使用，可以先建立一个临时的表 emp_null 并插入一条记录。在目前情况下读者不用理解这些建表和插入记录的 SQL 语句，而只需输入例 5-24 和例 5-25 的 SQL 语句即可。这些 SQL 语句会在后面讨论 DDL 语句的章节中详细介绍。

例 5-24

```
SQL> create table emp_null
  2  as select ename, sal, comm
  3  from emp;
```

例 5-24 结果

```
表已创建。
```

新建的 emp_null 表包含了 emp 表中的所有记录，但该表的每一行记录只有 3 列，分别为 ename、sal 和 comm。

例 5-25

```
SQL> insert into emp_null(ename, sal, comm)
  2  values ('QUEEN', NULL, NULL);
```

例 5-25 结果

```
已创建 1 行。
```

例 5-25 的 SQL 语句是向 emp_null 表中插入一条记录，该记录的 ename 为 QUEEN，sal 和 comm 都为空。现在您可以使用例 5-26 的查询语句来检验例 5-25 的插入操作是否正确。

例 5-26

```
SQL> SELECT * FROM emp_null;
```

例 5-26 结果

```
ENAME             SAL       COMM
---------- ---------- ----------
SMITH             800
ALLEN            1600        300
WARD             1250        500
JONES            2975
MARTIN           1250       1400
BLAKE            2850
CLARK            2450
SCOTT            3000
KING             5000
TURNER           1500          0
ADAMS            1100
JAMES             950
FORD             3000
MILLER           1300
QUEEN
已选择 15 行。
```

例 5-26 显示结果中的最后一行就是您刚插入的记录 QUEEN，该记录的 sal 和 comm 都为空。这在商业运作中的含义可能是：老板娘 QUEEN 是公司中唯一的一名义工（志愿者）。

现在您就可以使用例 5-27 的查询语句来测试 COALESCE 函数。

例 5-27

```
SQL> SELECT ename "Name", sal "Salary", comm "Commission",
  2         COALESCE(comm, sal*0.1, 100) "New Commission"
  3  FROM emp_null;
```

例 5-27 结果

```
Name           Salary Commission New Commission
---------- ---------- ---------- --------------
SMITH             800                        80
ALLEN            1600        300            300
WARD             1250        500            500
JONES            2975                     297.5
MARTIN           1250       1400           1400
BLAKE            2850                       285
CLARK            2450                       245
SCOTT            3000                       300
KING             5000                       500
```

TURNER	1500	0	0
ADAMS	1100		110
JAMES	950		95
FORD	3000		300
MILLER	1300		130
QUEEN			100

已选择 15 行。

从例 5-27 的显示结果中可以看出 COALESCE 函数的功能如下：

- 当 comm 不为空时，COALESCE 函数返回 comm 的值。例如，ALLEN 的佣金（comm）为 300，New Commission 的显示也为 300。
- 当 comm 为空且 sal*0.1 不为空（即 sae 不为空）时，COALESCE 函数返回 sal*0.1 的值。例如，SMITH 的佣金（comm）为空，但 sal=800 不为空，所以 COALESCE 函数返回 800*0.1=80。
- 当 comm 为空且 sal*0.1 也为空（即 sae 为空）时，COALESCE 函数返回 100。如 QUEEN 的佣金（comm）为空，并且 sal 也为空，所以 COALESCE 函数返回 100。

COALESCE 函数的格式如下：

COALESCE(表达式 1，表达式 2，表达式 3, …，表达式 n)

该函数返回表达式列表(表达式 1，表达式 2，表达式 3, …，表达式 n)中第 1 个不为空的表达式的值。

CASE 表达式也是 Oracle 9i 刚引入的，它的功能与 DECODE 函数完全相同，只是它的语法更类似于一般的程序设计语言的 CASE 语句。例 5-28 的查询语句与例 5-19 完成的功能完全相同，也许例 5-28 对程序员来说比较容易理解。

例 5-28

```
SQL> SELECT ename "Name", job, sal "Salary",
          CASE job WHEN 'SALESMAN' THEN sal*1.15
                   WHEN 'CLERK'    THEN sal*1.20
                   WHEN 'ANALYST'  THEN sal*1.25
                   ELSE                 sal*1.40  END "New Salary"
     FROM emp
     ORDER BY job;
```

例 5-28 结果

Name	JOB	Salary	New Salary
SCOTT	ANALYST	3000	3750
FORD	ANALYST	3000	3750
SMITH	CLERK	800	960
ADAMS	CLERK	1100	1320
MILLER	CLERK	1300	1560
JAMES	CLERK	950	1140
JONES	MANAGER	2975	4165

```
CLARK        MANAGER       2450       3430
BLAKE        MANAGER       2850       3990
KING         PRESIDENT     5000       7000
ALLEN        SALESMAN      1600       1840
MARTIN       SALESMAN      1250     1437.5
TURNER       SALESMAN      1500       1725
WARD         SALESMAN      1250     1437.5
已选择 14 行。
```

在例 5-28 的查询语句中，CASE 表达式执行的方法如下：

（1）当 JOB 为 SALESMAN 时，CASE 表达式返回表达式 sal*1.15 的值。

（2）否则，当 JOB 为 CLERK 时，CASE 表达式返回表达式 sal*1.20 的值。

（3）否则，当 JOB 为 ANALYST 时，CASE 表达式返回表达式 sal*1.25 的值。

（4）否则，CASE 表达式返回表达式 sal*1.40 的值。

5.11　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 什么是空值（NULL）？
- 引入空值（NULL）的原因。
- 空值（NULL）对表达式中运算的影响。
- 空值（NULL）是如何排序的？
- 什么是逻辑表达式？
- 3 个逻辑运算符。
- 含有空值（NULL）时 AND 逻辑运算符的真值表。
- 含有空值（NULL）时 OR 逻辑运算符的真值表。
- 含有空值（NULL）时 NOT 逻辑运算符的真值表。
- 运算符的优先级。
- 如何构造包含逻辑表达式的限制条件？
- NVL 函数。
- DECODE 函数。
- 单值函数的嵌套。

☠ **注意：**

以下的内容为 Oracle 9i 增加的新功能。如果您现在理解起来有困难的话，可以先跳过这部分，等您熟练掌握了 SQL 语句以后再重新看这部分的内容。

- NVL2 函数。
- NULLIF 函数。
- COALESCE 函数。
- CASE 表达式。

第6章

综合数据和分组函数

分组函数是对一批（一组）数据进行操作（综合）之后返回一个值，这批数据可能是整个表，也可能是按某种条件把该表分成的组。不少专家认为对于管理者或决策者来说，综合数据才是最有价值的信息。例如，对于一个大型企业的老总，他/她可能对企业的平均工资很感兴趣，但对该企业中每位员工的具体工资就没什么兴趣。

6.1　5个常用的分组函数

Oracle 常用的分组函数有 5 个，分别为 COUNT、AVG、SUM、MAX 和 MIN。下面分别进行介绍。

6.2　COUNT函数

COUNT 函数的语法格式如下：

COUNT({*[DISTINCT|ALL]表达式})

该函数用于返回查询的行数。

继续第 2 章开始时的例子。假设您的老板在决定把某人炒鱿鱼之前，想知道公司里到底有多少员工。您可以使用例 6-1 的查询语句来完成他的重托（在学习 Oracle 时，要想象使用 Oracle 数据库管理系统的公司规模可能很大，如该公司可能有一百多万名员工，这样就可以比较容易地理解所讨论的问题了）。

例 6-1

```
SQL> SELECT COUNT(*)
  2  FROM emp;
```

例 6-1 结果

```
  COUNT(*)
----------
        14
```

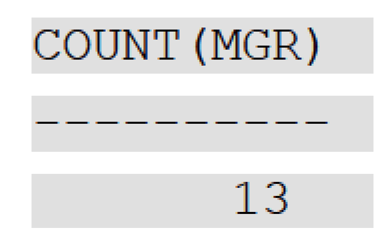
COUNT(*)用于返回表中所有的行，包括空行和重复的行。

如果老板想知道公司里有多少员工由经理管理（即不属于高级管理层），您可以使用例 6-2 的查询语句来完成他的这一重托。

例 6-2

```
SQL> SELECT COUNT(mgr)
  2  FROM emp;
```

例 6-2 结果



```
COUNT(MGR)
----------
        13
```

所谓经理管理的员工就是 mgr 不为空（NULL）的员工。因为 COUNT(表达式)返回表中所有表达式为非空的行，所以 COUNT(mgr) 返回表中所有 mgr 为非空的行，即由经理管理的员工的人数。

6.3　AVG和SUM函数

下面来讨论 AVG 和 SUM 函数。

AVG 函数的格式如下：

AVG([DISTINCT|ALL]表达式)

该函数用于返回表达式的平均值。

SUM 函数的格式如下：

SUM([DISTINCT|ALL]表达式)

该函数用于返回表达式的总合。

现在假设您的老板还想知道公司的员工的平均工资和工资总和，您就可以使用例 6-3 的查询语句来完成他的要求。

例 6-3

```
SQL> SELECT AVG(sal) "Average Salary", SUM(sal) "Summary", COUNT(sal) "Records"
  2  FROM emp;
```

例 6-3 结果

```
Average Salary    Summary    Records
-------------- ---------- ----------
    2073.21429      29025         14
```

在例 6-3 的查询语句中，您为每一列都给出了一个有意义的别名，这一点在工作中很重要，特别是当您的查询结果或报告拿给非计算机专业人员看时。在工作中要牢牢记住“客户永远是我们的上帝”。看报告的人就是客户，就是上帝。

6.4　MIN和MAX函数

现在来讨论 MAX 和 MIN 函数。

MAX 函数的格式如下：

MAX([DISTINCT|ALL]表达式)

该函数用于返回表达式的最大值。

MIN 函数的格式如下：

MIN([DISTINCT|ALL]表达式)

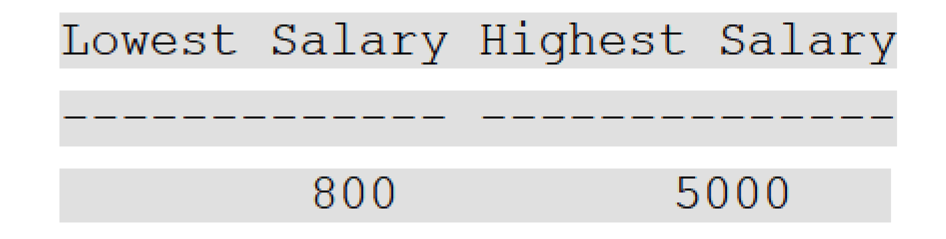
该函数用于返回表达式的最小值。

如果您的老板现在还想知道公司中员工的最低工资和最高工资，可以使用例 6-4 的查询语句来完成这一要求。

例 6-4

```
SQL> SELECT MIN(sal) "Lowest Salary", MAX(sal) "Highest Salary"
  2  FROM emp;
```

例 6-4 结果



```
Lowest Salary Highest Salary
------------- --------------
          800           5000
```

不像 AVG 和 SUM 函数只能操作数字型数据，MIN 和 MAX 函数不但可用于数字型数据，而且还可以用于字符型数据和日期型数据。例 6-5 的查询语句就是 MIN 和 MAX 函数用于字符型数据的一个例子。

例 6-5

```
SQL> SELECT MIN(job), MAX(job)
  2  FROM emp;
```

例 6-5 结果

```
MIN(job)  MAX(job)
--------- ---------
ANALYST   SALESMAN
```

假设老板又想知道公司雇用第一个员工和最后一个员工的日期，就可以使用例 6-6 的查询语句来完成这一新要求。

例 6-6

```
SQL> SELECT MIN(hiredate) "First Day", MAX(hiredate) "Last Day"
  2  FROM emp;
```

例 6-6 结果

```
First Day  Last Day
---------- ----------
17-12 月-1980 23-5 月-1987
```

在前面几节中，我们都是把一个表看成一个大组来处理。可以使用 GROUP BY 子句把一个表划分成若干个组，在一个表中建立多组数据。

6.5 GROUP BY子句的应用

如果老板现在又想知道公司中按职位（job）分类，每类员工的平均工资，可以用例 6-7 的查询语句来完成这一要求。

例 6-7

```
SQL> SELECT job, AVG(sal) "Average Salary"
    2  FROM emp
    3  GROUP BY job;
```

例 6-7 结果

```
job       Average Salary
--------- --------------
ANALYST             3000
CLERK             1037.5
MANAGER       2758.33333
PRESIDENT           5000
SALESMAN            1400
```

例 6-7 的结果显示除了总裁（president）以外，分析员（analyst）的工资最高。公司可以根据这一信息重组一些职位（job），例如，将分析员的一些简单的工作分给文员（clerk）来做，这样就可以解雇一些多余的分析员，从而为公司节省一些开销。

6.6 改变GROUP BY子句的排序次序

从例 6-7 的显示结果中可以看出，查询的结果是按 GROUP BY 子句中列的由小到大的顺序排列（升序排序）的，这也是使用 GROUP BY 子句时 Oracle 默认的排序方式。我们可以用 ORDER BY 子句来改变这一默认排序次序。

可以使用例 6-8 的查询语句来得到按职位（job）分类的每类员工的平均工资，并且显示的结果是按平均工资（Average Salary）由大到小排列（降序）。

例 6-8

```
SQL> SELECT job, AVG(sal) "Average Salary"
    2  FROM emp
    3  GROUP BY job
    4  ORDER BY "Average Salary" DESC;
```

例 6-8 结果

```
job       Average Salary
--------- --------------
PRESIDENT           5000
```

```
ANALYST            3000
MANAGER      2758.33333
SALESMAN           1400
CLERK            1037.5
```

例 6-8 不但证明了可以利用 ORDER BY 子句来改变 Oracle 默认的显示次序（使用 DESC 关键字），而且在 ORDER BY 子句中还可以使用组函数的别名（当然也可以使用组函数本身）。

6.7　GROUP BY子句的特殊用法

GROUP BY 子句中的列可以不在 SELECT 列表中。

可以使用例 6-9 的查询语句来得到按职位（job）分类（job 并没有包含在 SELECT 子句中）的每类员工的平均工资，并且显示的结果是按职位由小到大排列的（升序）。

例 6-9

```
SQL> SELECT AVG(sal) "Average Salary"
    2  FROM emp
    3  GROUP BY job;
```

例 6-9 结果

```
Average Salary
--------------
          3000
        1037.5
    2758.33333
          5000
          1400
```

从例 6-9 的显示结果中很难看出这一结果是按什么排序的。为了提高结果的可读性，应尽可能不使用这样的查询方法。

6.8　分组函数与GROUP BY子句的非法操作

如果使用例 6-10 的查询语句，Oracle 会返回错误信息。

例 6-10

```
SQL> SELECT job, AVG(sal)
2  FROM emp;
```

例 6-10 结果

```
SELECT job, AVG(sal)
       *
```

```
ERROR 位于第 1 行:
ORA-00937: 非单组分组函数
```

注意：

如果在一个查询中使用了分组函数，则任何不在分组函数中的列或表达式必须在 GROUP BY 子句中。

其实，如果仔细地研究一下例 6-10 的查询语句，就不难理解 Oracle 为什么会作出这样的规定。在 SELECT 子句中的 job 告诉 Oracle 系统显示每行数据的职位（job），在 emp 表中一共有 14 行的数据。而在 SELECT 子句中的 AVG(sal) 告诉 Oracle 系统显示 emp 表中所有数据行的平均工资，在这个查询语句中只能产生一个平均工资。查询语句的这两个要求显然是矛盾的，因此，Oracle 系统就不知道怎样去做。最简单的解决方案就是什么也不做，只报个错就可以了。

为了改正这一错误，您可以在例 6-10 的查询语句中增加 GROUP BY 子句，并把列 job 放在该子句中。修改后的语句如例 6-11 所示。

例 6-11

```
SQL> SELECT job, AVG(sal)
   2  FROM emp
   3  GROUP BY job;
```

例 6-11 结果

```
job          AVG(SAL)
--------- ----------
ANALYST         3000
CLERK         1037.5
MANAGER   2758.33333
PRESIDENT       5000
SALESMAN        1400
```

例 6-11 显示的结果给出了 emp 表中每种职位（job）的平均工资（AVG(sal)）。

例 6-12 是有关分组函数与 GROUP BY 子句非法操作的一个比较复杂的例子。

例 6-12

```
SQL> SELECT job, AVG(sal)
   2  FROM emp
   3  GROUP BY job
4  ORDER BY deptno;
```

请问例 6-12 的查询语句中是否有错误？如果有错误，错在什么地方?

例 6-12 的查询语句是错的，错在 ORDER BY 子句上。因为 deptno 既不在分组函数中，也不在 GROUP BY 子句中。下面是例 6-12 的显示结果。

例 6-12 结果

```
ORDER BY deptno
         *
```

```
ERROR 位于第 4 行:
ORA-00979: 不是 GROUP BY 表达式
```

6.9　HAVING子句的使用

如果老板只想知道平均工资高于 1500 元的职位（工种），该如何改写例 6-7 的查询语句呢？您可以使用例 6-13 的查询语句。

例 6-13

```
SQL> SELECT job, AVG(sal)
   2  FROM emp
   3  WHERE AVG(sal) > 1500
4  GROUP BY job;
```

例 6-13 结果

```
WHERE AVG(sal) > 1500
     *
ERROR 位于第 3 行:
ORA-00934: 此处不允许使用分组函数
```

例 6-13 显示的结果却让您有些失望，这是因为 WHERE 子句不能用于限制分组函数。在 Oracle 中可以使用 HAVING 子句来限制分组函数。现在您可以将例 6-13 的查询语句中的 WHERE 换成 HAVING，如例 6-14 所示。

例 6-14

```
SQL> SELECT job, AVG(sal)
   2  FROM emp
   3  HAVING AVG(sal) > 1500
   4  GROUP BY job;
```

例 6-14 结果

```
JOB         AVG(SAL)
--------- ----------
ANALYST         3000
MANAGER   2758.33333
PRESIDENT       5000
```

当使用了 HAVING 子句时，Oracle 系统处理的顺序如下：

（1）首先对数据行（记录）进行分组。

（2）把所得到的分组应用于分组函数。

（3）最后显示满足 HAVING 子句所指定条件的结果。

根据以上对 HAVING 子句的讨论，虽然例 6-14 的查询语句得到了所需要的结果，但是它与 Oracle 处理带有 HAVING 子句的查询语句的逻辑顺序不同。因此，最好把 HAVING 子

句放在 GROUP BY 子句之后。您可以将上述查询语句修改成如下的 SQL 语句（见例 6-15）。

例 6-15

```
SQL> SELECT job, AVG(sal)
  2  FROM emp
  3  GROUP BY job
  4  HAVING AVG(sal) > 1500
  5  ORDER BY 2;
```

例 6-15 结果

```
JOB          AVG(SAL)
--------- ----------
MANAGER    2758.33333
ANALYST          3000
PRESIDENT        5000
```

在实际工作中，例 6-15 中的 ORDER BY 2 应写成 ORDER BY AVG(sal)，这样的易读性更好。本书中之所以使用这种方式，是因为这种用法曾在 OCP（Oracle CERTIFIED PROFESSIONAL）考试中出现过，但是在有关 Oracle SQL 的书中却没有介绍过。其目的是为了让读者熟悉这种用法。

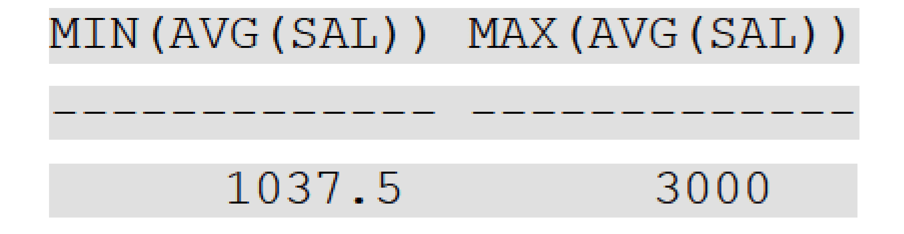
6.10 分组函数的嵌套

我们继续例 6-7 的讨论。公司高级管理层在进行优化组合之前想知道按职位（job）分类最低和最高平均工资（当然不能包括总裁，因为他要领导该企业的重组）。他们可以根据这些信息估算出“将最高平均工资职位（job）的一些简单的工作分给最低平均工资职位（job）的员工来做；之后解雇一些多余的最高平均工资职位（job）的员工，保留或增加最低平均工资职位（job）的员工”。这一重大的企业重组可为公司节省多少钱？可以使用例 6-16 的查询来得到他们所需要的信息。

例 6-16

```
SQL> SELECT MIN(AVG(sal)), MAX(AVG(sal))
  2  FROM emp
  3  WHERE job NOT LIKE 'PRESI%'
  4  GROUP BY job;
```

例 6-16 结果



```
MIN(AVG(SAL)) MAX(AVG(SAL))
------------- -------------
       1037.5          3000
```

由例 6-16 显示的结果看出，在公司中最高平均工资约是最低平均工资的 3 倍。因此，若能解雇一些最高平均工资职位（job）的员工，而把最高平均工资职位（job）的一些工作分给最低平均工资职位（job）的员工来做，应该能节省不少开销。

在例 6-16 的查询语句中我们使用了两层的分组函数的嵌套。在 Oracle 系统中嵌套的分组函数计算顺序是由内到外的，也就是说，在 Oracle 系统中按如下的顺序来执行例 6-16 的查询语句。

（1）在 emp 表中找到所有职位（job）不是以 PRESI 开头的数据行（记录）。

（2）将这些数据行（记录）按职位（job）分类。

（3）求出每一类的平均工资。

（4）最后求出这些平均工资的最小值和最大值。

与单值函数不同，分组函数只能嵌套两层。

注意：

尽管分组函数给我们编写 SQL 语句带来了很大的方便，但是使用起来可能会使系统的效率明显下降，特别是在对容量大的表格进行这样的操作时。因为使用分组函数通常要扫描整个表，如果使用了 GROUP BY 子句，Oracle 还要进行排序。

6.11　分组函数的空值问题

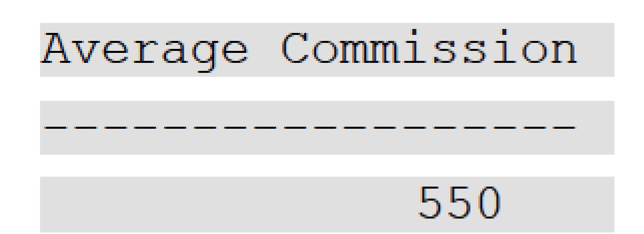
除了 COUNT(*)以外，其他的分组函数都不处理空值（NULL）。

如果您想得到公司员工的平均佣金，可以使用例 6-17 的查询语句。

例 6-17

```
SQL> SELECT AVG(comm) "Average Commission"
   2  FROM emp;
```

例 6-17 结果



```
Average Commission
------------------
               550
```

但是例 6-17 的结果似乎有些问题，因为所显示的平均佣金明显偏高。其原因是 AVG(comm)不包括 comm 为空值的记录行，而在整个公司中只有推销员（SALESMAN）有佣金。所以，例 6-17 显示的结果其实是推销员（SALESMAN）的平均佣金。可以使用例 6-18 的 SQL 语句来验证我们的分析。

例 6-18

```
SQL> SELECT AVG(comm) "Average Commission", SUM(comm) "Summary Commission",
   2         job, COUNT(comm) "Records"
   3  FROM emp
4  GROUP BY job;
```

例 6-18 结果

```
Average Commission Summary Commission job        Records
------------------ ------------------ ---------- ----------
                                      ANALYST          0
```

		CLERK	0
		MANAGER	0
		PRESIDENT	0
550	2200	SALESMAN	4

从例 6-18 的显示结果可以看出，Oracle 先求出佣金的总数，再除以挣佣金的员工总人数（推销员的总人数），即 2200/(4+0+0+0+0)= 550。

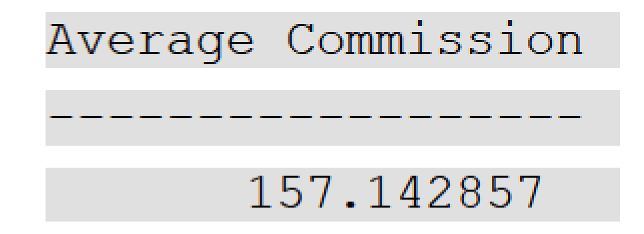
6.12　NVL函数在分组函数中的使用

如果想让所有的员工都参加平均，可借助于 NVL 函数。可以将例 6-17 的查询语句修改成例 6-19 的 SQL 语句。

例 6-19

```
SQL> SELECT AVG(NVL(comm, 0)) "Average Commission"
   2  FROM emp;
```

例 6-19 结果



```
Average Commission
------------------
        157.142857
```

例 6-19 的显示结果似乎就是我们所期望的员工的平均佣金。可以使用例 6-20 的 SQL 语句来验证我们的猜测。

例 6-20

```
SQL> SELECT AVG(NVL(comm, 0)) "Average Commission",
2        SUM(NVL(comm, 0)) "Summary Commission",
   3        job, COUNT(NVL(comm, 0)) "Records"
   4  FROM emp
5  GROUP BY job;
```

例 6-20 结果

Average Commission	Summary Commission	job	Records
0	0	ANALYST	2
0	0	CLERK	4
0	0	MANAGER	3
0	0	PRESIDENT	1
550	2200	SALESMAN	4

例 6-20 的结果表明这一平均佣金就是我们所要的结果，因为(2200+0+0+0+0)/(2+4+3+1+4)=157.142871。

6.13　是否在分组函数中使用NVL函数的商业背景

读者也许会问，在实际商业运作中，什么时候需要使用例 6-17 的查询语句，即不使用 NVL 函数呢？什么时候需要使用例 6-19 的查询语句，即使用 NVL 函数呢？

以下假想的情形可能会对您作出正确的选择有所帮助。

如果是为公司的股东大会准备报告，公司的高级管理层也许更喜欢像例 6-19 的查询语句，因为这样看上去公司付给员工的佣金很低，会让股东们开心，可能会帮助吸引更多的投资。如果公司是在招收新的推销员，公司经理正在向应征者介绍推销员的佣金水平，公司的经理们也许更喜欢例 6-17 的查询语句，因为较高的佣金水平可以更容易地吸引一些优秀的推销员。

6.14　其他的分组函数和分组函数的小结

Oracle 常用的分组函数还有 STDDEV 和 VARIANCE。有关这两个函数的讨论已超出了本书的范围，如果读者对这两个分组函数感兴趣，可参阅 Oracle 9i SQL Reference 6-139（337 页）和 6-195（393 页）中有关标准方差的公式。

如果在一个查询中使用了分组函数，任何不在分组函数中的列或表达式必须在 GROUP BY 子句中。如果在一个查询中所使用的限制条件中包括了分组函数，则该限制条件必须放在 HAVING 子句中而不能放在 WHERE 子句中。MIN 和 MAX 函数不但可用于数字型数据，而且还可用于字符型数据和日期型数据。分组函数在决策支持系统（DSS：Decision Support System）中使用得比较多。虽然分组函数为管理者或决策者提供了丰富的信息，但它们对系统效率的冲击是不容忽视的。

6.15　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 什么是分组函数？
- 5 个常用的分组函数的用法。
- MIN 和 MAX 函数是如何用于字符型数据和日期型数据的？
- 如何利用 GROUP BY 子句把一个表划分成若干个组？
- GROUP BY 子句的排序问题。
- 如何避免分组函数与 GROUP BY 子句的非法操作？
- 如何利用分组函数来限制查询语句的结果？
- 分组函数的嵌套。
- 分组函数中的空值处理。
- 分组函数中空值处理的商业背景。

第7章

多表查询

在前面各章所使用的例子中，每个查询语句只对一个表操作。其实在 Oracle 系统中，您所查询的数据可以来自多个表，即一个查询语句可以对多个表操作。造成这一现象的原因主要是由于数据库的规范化（Normalization）。那么什么是数据库的规范化呢？以及为什么要进行数据库的规范化？下面我们就作一个非常简单的介绍。

7.1　数据库的规范化

在介绍多表查询之前，我们先讨论一下产生多表查询的主要原因。图 7.1 是一张简化了的公司订单。其公司的名字为“天上人间快餐大中华总店”。

供应商号：234510
供应商：神州商霸批发有限公司
地址：　XX 省，XX 市，XX 区
财源道 666 号，174 号楼
地下防空洞

日期：　XXXX 年 XX 月 XX 日
订单号：　168268

商品号：	商品名	数　量	单　价（元）	合　计（元）
2560	清凉牌一次性纸杯	1000	0.20	200.00
2351	龙卷风牌一次性筷子	5000	0.10	500.00
2354	万里香牌一次性饭盒	1000	0.20	200.00
2546	好运来牌餐巾纸	100（盒）	1.00	100.00
2200	一滴灵牌洗净剂	20（瓶）	2.00	40.00

总　计：　1，040.00

天上人间快餐大中华总店
XX 省，XX 市，XX 区
进宝路 368 号地下室
电话：666 1744
传真：666 9444

图　7.1

如何基于这张订单来设计一个关系数据库的表呢？一种最简单的办法是把该订单的所有数据存放在一个表中，并用订单号和商品号做主键（Primary Key），如图 7.2 所示。

订单号	商品号	商品名	商品描述	单 价	……	供应商号	供应商名	……
168268	2560	清凉牌一次性纸杯	……	0.20	……	234510	……	
	2351	龙卷风牌一次性筷子	……	0.10	……	……	……	
	2354	万里香牌一次性饭盒	……	0.20	……	……	……	
	2546	好运来牌餐巾纸	……	1.00	……	……	……	
	2200	一滴灵牌洗净剂	……	2.00	……	……	……	
168269	2560	清凉牌一次性纸杯	……	0.20		234510	……	
	2351	龙卷风牌一次性筷子	……	0.10		……	……	
	2354	万里香牌一次性饭盒		0.20				
168288	2351	龙卷风牌一次性筷子	……	0.10		234521	……	
	2546	好运来牌餐巾纸	……	1.00		……	……	
	2200	一滴灵牌洗净剂	……	2.00		……	……	

图　7.2

7.2　主键和实体完整性

图 7.2 并不是一个关系数据库的表，因为所有的关系数据库表中的每一行必须有一主键（Primary Key），并且要满足实体完整性（Entity Integrity）。图 7.2 中的阴影部分叫重复组（Repeating Groups）或重复属性。关系数据库表中是不能含有重复组的。

那么什么样的表是关系数据库的表呢？在回答这一问题之前，我们先介绍两个在关系数据库中非常重要的概念：主键（Primary Key）和实体完整性（Entity Integrity）。

- 主键（Primary Key）：它是关系数据库表中的某一列或某几列的组合；它能够唯一地标识关系数据库表中的任一行。
- 实体完整性（Entity Integrity）：主键不能包含空值（NULL），并且主键必须能够唯一地标识任一行。

表的设计者负责定义主键，关系数据库管理系统（如 Oracle）负责维护实体完整性。

根据实体完整性的定义，图 7.2 显然不是一个关系数据库的表，因为在这个表中有些行的主键部分是空的。

7.3　第 一 范 式

我们可以对图 7.2 做一些修改，使之变成关系数据库的表（Order），如图 7.3 所示。

图 7.3 已经是一个关系数据库表。您现在能看出该表在实际应用中的问题吗？

假设商品号为 2351 的龙卷风牌一次性筷子是天上人间快餐大中华总店经常订的商品，而天上人间快餐大中华总店已发展成一个跨地域的连锁店，该公司每年可能要订购 2351 号商品上百次。也就是说 2351 号商品的信息要在 Order 表中重复上百次。这样出现输入错误

的可能性大大增加，从而产生相同的商品在同一表中不同的行中记录的信息不同，也就是所谓的数据不一致。

数据的不一致性是由于相同数据的重复存放，即冗余造成的。数据的冗余不但会造成数据不一致，还会使系统的维护更加困难，也会对系统的效率产生冲击。

如何减少数据的冗余呢？实际上图 7.3 所示的 Order 表是一个第一范式（1NF）的表。任何关系数据库的表都满足第一范式（1NF）。

第一范式的定义如下：

（1）所有的键属性（列）都已定义。

（2）没有任何重复组（Repeating Groups），换句话说，每行和每列的交汇处可以而且只能包含一个值，而不能包含一组值。

（3）所有的属性（列）都依赖于主键。

订单号	商品号	商品名	商品描述	单 价	……	供应商号	供应商名	……
168268	2560	清凉牌一次性纸杯	……	0.20		234510	……	
168268	2351	龙卷风牌一次性筷子	……	0.10	……	……	……	
168268	2354	万里香牌一次性饭盒	……	0.20	……	……	……	
168268	2546	好运来牌餐巾纸	……	1.00	……	……	……	
168268	2200	一滴灵牌洗净剂	……	2.00	……	……	……	
168269	2560	清凉牌一次性纸杯	……	0.20		234510	……	
168269	2351	龙卷风牌一次性筷子	……	0.10		……	……	
168269	2354	万里香牌一次性饭盒		0.20				
168288	2351	龙卷风牌一次性筷子	……	0.10		234521	……	
168288	2546	好运来牌餐巾纸	……	1.00		……	……	
168288	2200	一滴灵牌洗净剂	……	2.00		……	……	
……								

图 7.3

7.4 消除部分依赖

仔细分析图 7.3 所示的 Order 表您就会发现，我们只要知道了某一商品的商品号就能知道该商品的全部信息，即商品其他部分的信息只依赖于商品号。图 7.4 为表示此关系的依赖关系图。

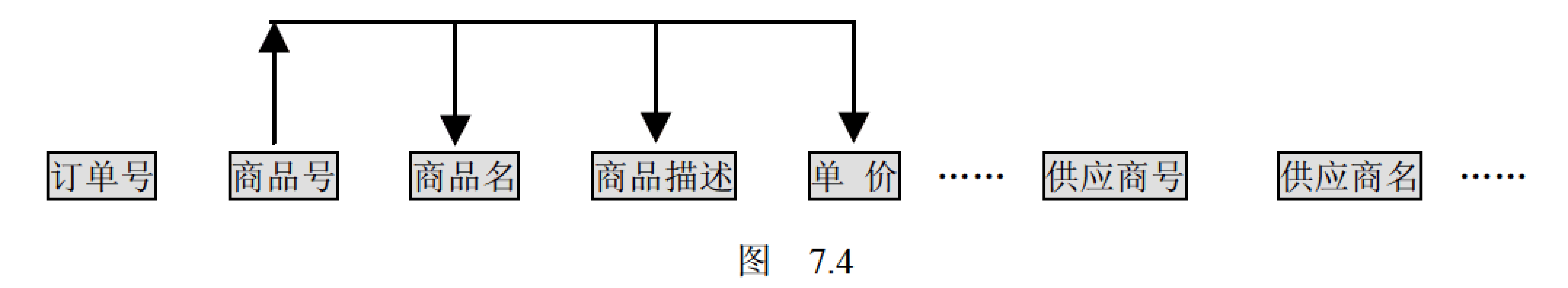


图 7.4

依赖关系图是一个帮助我们找到依赖关系的非常有效的工具。由图 7.4 可以很清楚地

看出，商品所有其他部分的信息都依赖于商品号。因为商品号只是主键的一部分，所以这种依赖关系也叫部分依赖。

部分依赖（Partial Dependency）是指只依赖于部分主键的依赖关系。

我们可以把存在着部分依赖关系的列拿出来重新生成一个新表叫 Product，而把商品号作为新表 Product 的主键，同时把商品号也保留在新的 Order 表中作为外键（稍后我们将介绍什么是外键）。

现在我们把一个表（Order）拆分成了两个表（Order 和 Product），如图 7.5 和图 7.6 所示，它们之间通过商品号来连接。



图　7.5

图　7.6

现在一个商品的信息只需在 Product 表中存放一次，而 Product 表中只有商品号在 Order 表中可重复存放多次，因此，产生的数据的冗余已经大大地减少。

7.5　外键和引用完整性

那么关系数据库管理系统又是如何从多个表中找到用户所需的信息呢？使关系数据库工作的机制是控制冗余。

控制冗余就是数据库中的表通过共享相同的键属性（列）把它们链接在了一起。

例如，Order 表和 Product 表就是靠共享商品号链接在一起的。商品号在 Product 表中是主键，而它在 Order 表中为外键。

外键（Foreign Key）是关系数据库表中的某一列或某几列的组合；它的值或者与另一个表中（有时也可能是同一个表中）的某一列（一般为主键（Primary Key））相匹配，或者为空值（NULL）。

引用完整性（Referential Integrity）的定义如下：

（1）外键必须为空值（NULL）或者有相匹配的项。

（2）外键可以没有相对应的键属性（列），但不可以有无效的项。

表的设计者负责定义外键（Foreign Key）。关系数据库管理系统（如 Oracle）负责维护引用完整（Referential Integrity）。

7.6　第 二 范 式

消除重复组（Repeating Groups）和部分依赖的表为第二范式（2NF）的表，因此图 7.5

（Order）所示的表已经是第二范式的表。

第二范式的定义如下：

（1）该表为第一范式（1NF）的表。

（2）该表不包含部分依赖。

如果一个表是第一范式（1NF）的表，并且它的主键由一列（单一属性）组成，那么该表自动地成为第二范式（2NF）的表。

我们继续分析图 7.5（Order）。

7.7 第 三 范 式

假设供应商——神州商霸批发有限公司是一个非常好的供应商，天上人间快餐大中华总店每年要向该公司订购上百次商品。也就是说 234510 号供应商的信息要在 Order 表中重复上百次。

现在进一步假设：虽然该供应商在起家时，只不过是在地下防空洞中经营的一个夫妻店，但经过夫妻俩艰苦奋斗和有效经营，生意越来越红火。现在已在本市的商业中心区租下了几层楼，电话也改了，且又增加了 8 条线，而且又雇了个经理和一大帮推销员。两人已不再参与公司的日常管理了。

虽然这些变化与天上人间快餐大中华总店毫不相干，但由于它的数据库的设计有缺陷，所以这些变化都可能影响到 Order 表，即可能要修改这个表；而且一个变化可能引起几百行数据的修改。

如何来解决这一问题呢？我们还是先用依赖关系图来分析各个列之间的依赖关系，如图 7.7 所示。

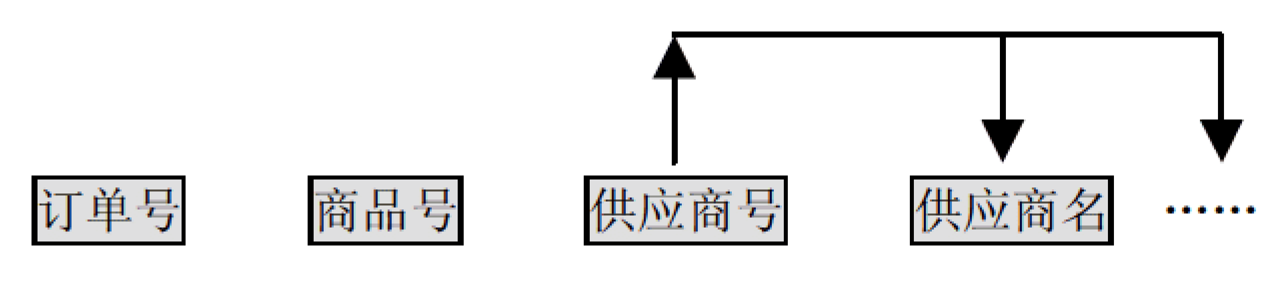


图 7.7

从图 7.7 可以看出，我们只要知道了某一供应商的供应商号就能知道该供应商的全部信息，即供应商的其他部分的信息只依赖于供应商号。但供应商号不是主键，因此它不属于部分依赖。那么它属于哪种依赖呢？

传递依赖（Transitive Dependency）是指一个或多个属性（列）依赖于非主键的属性（列）。

我们再把存在着传递依赖关系的列拿出来重新生成一个新表叫 Supplier，而把供应商号作为新表 Supplier 的主键，同时把供应商号保留在新的 Order 表中作为外键。这样就把最初的一个表分拆成了如图 7.8、图 7.9、图 7.10 所示的 3 个表。

图 7.8 Order、图 7.9 Supplier 和图 7.10 Product 中的每一个表都有主键并且不包含重复组（Repeating Group），同时也不包含部分依赖和传递依赖，这就是我们所说的第三范式（3NF）。

订单号	商品号	供应商号
168268	2560	234510
168268	2351	234510
168268	2354	234510
168268	2546	234510
168268	2200	234510
168269	2560	234510
168269	2351	234510
168269	2354	234510
168288	2351	234521
168288	2546	234521
168288	2200	234521
……		

图 7.8

供应商号	供应商名	……
234510	神州商霸批发有限公司	
234521	环宇无敌商业有限公司	
234530	心太软小商品批发有限公司	
……		

图 7.9

商品号	商品名	商品描述	单　价	……
2560	清凉牌一次性纸杯	……	0.20	……
2351	龙卷风牌一次性筷子	……	0.10	……
2354	万里香牌一次性饭盒	……	0.20	……
2546	好运来牌餐巾纸	……	1.00	……
2200	一滴灵牌洗净剂	……	2.00	……
……				

图 7.10

第三范式的定义如下：

（1）该表为第二范式（2NF）的表。

（2）该表不包含传递依赖。

对于绝大多数商业数据库设计来说，第三范式（3NF）就是规范化过程的终点。

7.8　规范化过程小结

规范化是一个过程，该过程决定把哪些属性（列）放在哪些实体（表）中。

规范化可以帮助我们设计好的表结构，因为规范化减少了数据的冗余，这使得系统更容易维护而且系统效率也会更高。

所谓的关系型数据库的逻辑设计就是数据库的表结构设计。在关系型数据库中，只有表中存有数据。

规范化只减少了数据的冗余而并没有完全消除冗余。因为为了使相关的表连接起来，在规范化过程中不得不引入控制冗余（外键）。不过这种控制冗余与第一范式（1NF）和第二范式（2NF）表的数据冗余相比已经少得微不足道了。

注意：

本书所介绍的数据库的规范化只是简单概括，其目的是为了帮助读者理解以后章节的一些内容，如多表连接、DML 操作等。

7.9 多 表 连 接

为了减少数据的冗余，商业数据库的设计者使用规范化过程来设计表的结构。对于一个好的数据库设计来说，其中绝大多数的表为第三范式（3NF）。但这也产生了一个问题，即原来在商业上一起使用的数据现在被放到了若干个不同的表中（如前面讨论的订单系统）。如何在多个表中得到所需的信息呢？Oracle 是用连接（Join）来完成多表查询的。Oracle 提供了 4 种类型的连接，分别为相等连接（Equi join）、自连接（Self join）、不等连接（Non-equijoin）和外连接（Outer join）。下面分别介绍这几种类型的连接。

7.10 相 等 连 接

假设您的公司高级管理层在进行优化组合之前想知道每个员工所属部门和所在的具体地点。您可以使用例 7-1 的查询语句。

例 7-1

```
SQL> SELECT empno, ename, sal, emp.deptno, loc
  2  FROM emp, dept
  3  WHERE emp.deptno = dept.deptno
  4  ORDER BY loc;
```

例 7-1 结果

```
     EMPNO ENAME             SAL     DEPTNO LOC
---------- ---------- ---------- ---------- ---------
      7499 ALLEN            1600         30 CHICAGO
      7521 WARD             1250         30 CHICAGO
      7654 MARTIN           1250         30 CHICAGO
      7900 JAMES             950         30 CHICAGO
      7844 TURNER           1500         30 CHICAGO
      7698 BLAKE            2850         30 CHICAGO
      7369 SMITH             800         20 DALLAS
      7902 FORD             3000         20 DALLAS
      7876 ADAMS            1100         20 DALLAS
```

```
      7566 JONES            2975          20 DALLAS
      7788 SCOTT            3000          20 DALLAS
      7782 CLARK            2450          10 NEW YORK
      7839 KING             5000          10 NEW YORK
      7934 MILLER           1300          10 NEW YORK
已选择 14 行。
```

例 7-1 为相等连接，即当两个表中记录的 deptno 值完全相等时才进行连接。通常这种连接查询涉及主键和外键。相等连接也称简单连接或内连接。以下是使用连接的一些指南，这些指南不但适合于相等连接而且也适合于其他类型的连接：

- 要连接的表都要放在 FROM 子句中，表名用逗号分开（如 FROM emp,dept）。
- 连接的条件放在 WHERE 子句中（如 WHERE emp.deptno = dept.deptno）。
- 要进行 n 个表的连接需要至少 n−1 个连接条件。
- 如果多个表中有相同列名的列时，在这些列的前面要冠以表名来区别它们，表名和列名之间用逗号隔开。
- 在列前冠以表名可以改善系统的效率，因为表名告诉 Oracle 服务器到哪一个表中找哪些列。

如果公司高级管理层只想知道工资为 1500 元或以上的员工所属部门和所在的具体地点，您可以使用例 7-2 的查询语句。

例 7-2

```
SQL> SELECT emp.empno, emp.ename, emp.sal,
   2        emp.deptno, dept.loc
   3  FROM emp, dept
   4  WHERE emp.deptno = dept.deptno
   5  AND sal >= 1500
6  ORDER BY loc;
```

例 7-2 结果

```
     EMPNO ENAME             SAL     DEPTNO LOC
---------- ---------- ---------- ---------- --------
      7499 ALLEN            1600          30 CHICAGO
      7698 BLAKE            2850          30 CHICAGO
      7844 TURNER           1500          30 CHICAGO
      7566 JONES            2975          20 DALLAS
      7902 FORD             3000          20 DALLAS
      7788 SCOTT            3000          20 DALLAS
      7782 CLARK            2450          10 NEW YORK
      7839 KING             5000          10 NEW YORK
已选择 8 行。
```

从例 7-2 可以看出，除了连接条件外，我们还可以在 WHERE 子句中加入其他的条件，这些条件由 AND 运算符组合起来。

7.11　连接中表别名的使用

从例 7-2 还可以看出，如果一个表的名字很长（如 INVOICE-DETAILS），在列前冠以表名需要大量的输入。Oracle 提供了一种简便的方法，就是使用表的别名。您可以使用表的别名将例 7-2 查询语句修改成例 7-3 的查询语句。

例 7-3

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno, d.loc
  2  FROM emp e, dept d
  3  WHERE e.deptno = d.deptno
  4  AND e.sal >= 1500
  5  ORDER BY d.loc;
```

例 7-3 结果

```
     EMPNO ENAME             SAL     DEPTNO LOC
---------- ---------- ---------- ---------- --------
      7499 ALLEN            1600         30 CHICAGO
      7698 BLAKE            2850         30 CHICAGO
      7844 TURNER           1500         30 CHICAGO
      7566 JONES            2975         20 DALLAS
      7902 FORD             3000         20 DALLAS
      7788 SCOTT            3000         20 DALLAS
      7782 CLARK            2450         10 NEW YORK
      7839 KING             5000         10 NEW YORK
已选择 8 行。
```

以下是使用表的别名的一些指南：

- 表的别名在 FROM 子句中定义，别名放在表名之后，它们之间用空格隔开。
- 别名一经定义，在整个的查询语句中就只能使用表的别名而不能再使用表名。
- 表的别名只在所定义的查询语句中有效。
- 应该选择有意义的别名，表的别名最长为 30 个字符，但越短越好。

7.12　笛卡儿乘积（乘积连接）

如果您在例 7-3 的查询语句中忘记了 WHERE 子句，那么会得到什么样的结果呢？请看例 7-4 的查询语句。

例 7-4

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno, d.loc
  2  FROM emp e, dept d
  3  ORDER BY d.loc;
```

例 7-4 结果

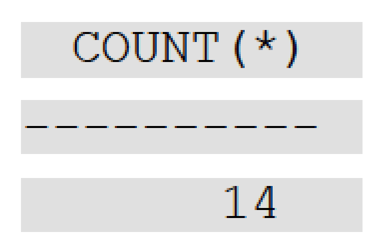
```
    EMPNO ENAME             SAL     DEPTNO LOC
---------- ---------- ---------- ---------- --------
     7369 SMITH             800         20 BOSTON
     7499 ALLEN            1600         30 BOSTON
     7521 WARD             1250         30 BOSTON
     7698 BLAKE            2850         30 BOSTON
     7782 CLARK            2450         10 BOSTON
     7654 MARTIN           1250         30 BOSTON
     7566 JONES            2975         20 BOSTON
     7788 SCOTT            3000         20 BOSTON
     7844 TURNER           1500         30 BOSTON
     7934 MILLER           1300         10 BOSTON
     7902 FORD             3000         20 BOSTON
     7900 JAMES             950         30 BOSTON
     7876 ADAMS            1100         20 BOSTON
     7839 KING             5000         10 BOSTON
     7369 SMITH             800         20 CHICAGO
     7499 ALLEN            1600         30 CHICAGO
     7521 WARD             1250         30 CHICAGO
     7566 JONES            2975         20 CHICAGO
     7788 SCOTT            3000         20 CHICAGO
     7844 TURNER           1500         30 CHICAGO
     7900 JAMES             950         30 CHICAGO
…
     7844 TURNER           1500         30 NEW YORK
     7876 ADAMS            1100         20 NEW YORK
     7900 JAMES             950         30 NEW YORK
     7902 FORD             3000         20 NEW YORK
     7934 MILLER           1300         10 NEW YORK
已选择 56 行。
```

例 7-4 的结果显示了 56 行的记录。这是如何得到的呢？我们用例 7-5 和例 7-6 的查询语句来分别求出 emp 表和 dept 表的记录数。

例 7-5

```
SQL> SELECT COUNT(*)
  2  FROM emp;
```

例 7-5 结果

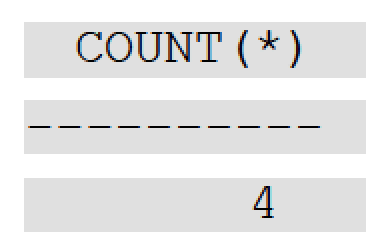


```
  COUNT(*)
----------
        14
```

例 7-6

```
SQL> SELECT COUNT(*)
   2  FROM dept;
```

例 7-6 结果



```
 COUNT(*)
----------
         4
```

例 7-5 和例 7-6 的结果显示 emp 有 14 条记录，dept 有 4 条记录，因此，例 7-4 显示的结果为 14×4=56 行的记录。笛卡儿乘积（乘积连接）的结果包括了所有表中的所有行的组合。

也许读者并不觉得例 7-4 的查询语句会有太大的问题。但是在一个大型公司的数据库中的一些表可能非常大，如果求一个有一百万行数据的表和另一个有一万行数据的表的笛卡儿乘积，其结果就为：1000000×10000=10000000000 行。这样的查询会对系统的效率产生极大的冲击，但查询的结果却几乎没什么用处。

以下是笛卡儿乘积（乘积连接）形成的条件：

- 查询语句中漏掉了连接条件（join condition）。
- 查询语句中两个表中的所有行都满足连接条件（join condition）。
- 查询语句中的连接条件无效。

在实际工作中要尽可能地避免产生笛卡儿乘积（乘积连接）。要达到这一目标，在多表连接查询语句的 WHERE 子句中，必须永远使用有效而正确的连接条件。

7.13 自 连 接

如果公司高级管理层现在就要准备实施他们的优化组合方案，即先解雇一些高工资的分析员。在实施优化组合方案之前，他们要找每一位分析员的头（经理）谈话以确定合适的人选。下面的查询语句可以得到高级管理层所需的信息。

例 7-7

```
SQL> SELECT w.empno, w.ename, w.job, w.mgr, m.ename "M_Name", m.job "M_Job"
   2  FROM emp w, emp m
   3  WHERE w.mgr = m.empno
4  AND w.job LIKE 'ANA%';
```

例 7-7 结果

```
     EMPNO ENAME      JOB              MGR M_Name     M_Job
---------- ---------- --------- ---------- ---------- -------
      7788 SCOTT      ANALYST         7566 JONES      MANAGER
      7902 FORD       ANALYST         7566 JONES      MANAGER
```

Oracle 是通过把一个表定义了两个不同的别名的方法（即把一个表映射成两个表）来

完成自连接（Self join）的。实际上我们可以不使用自连接（Self join），而通过例 7-8 和例 7-9 这两个简单的查询来得到与例 7-7 完全相同的结果。

例 7-8

```
SQL> SELECT ename, job, mgr
  2  FROM emp
  3  WHERE job LIKE 'ANA%';
```

例 7-8 结果

```
ENAME      JOB              MGR
---------- --------- ----------
SCOTT      ANALYST         7566
FORD       ANALYST         7566
```

例 7-9

```
SQL> SELECT empno, ename, job
  2  FROM emp
  3  WHERE empno = 7566;
```

例 7-9 结果

```
     EMPNO ENAME      JOB
---------- ---------- ---------
      7566 JONES      MANAGER
```

也许有人会觉得如果把所有经理的信息放在一个单独的表中查询可能会更容易些。也许这种想法不错。我们现在不妨来试一试。先创建一个 MANAGER 表，为了简单起见，MANAGER 表的内容与 EMP 表中的一模一样（在目前阶段读者不需要理解创建表命令的含义，而只需要按照例 7-10 的语句输入即可）。

例 7-10

```
SQL> CREATE TABLE manager
  2  AS
  3  SELECT *
  4  FROM emp;
```

例 7-10 结果

```
表已创建。
```

例 7-11 是使用新建的表 manager 和表 emp 的一个相等连接的查询语句，其结果与例 7-7 自连接（Self join）的查询完全一样。

例 7-11

```
SQL> SELECT w.empno, w.ename, w.job, w.mgr, m.ename "M_Name", m.job "M_Job"
  2  FROM emp w, manager m
  3  WHERE w.mgr = m.empno
  4  AND w.job LIKE 'ANA%';
```

例 7-11 结果

```
EMPNO  ENAME        JOB            MGR  M_Name      M_Job
---------- ---------- ---------- ---------- ---------- --------
      7788  SCOTT        ANALYST        7566  JONES      MANAGER
      7902  FORD         ANALYST        7566  JONES      MANAGER
```

尽管例 7-11 的查询结果与例 7-7 自连接（Self join）的查询结果完全一样，但这样做需要多创建一个表 manager，而表 manager 的内容全部来自表 emp。也就是说，表 emp 中任何记录的修改都有可能引起表 manager 中相应记录的修改。因此，这样的系统是很难维护的。

7.14 两个以上的表的连接

因为解雇员工是一件非常重大的决定，公司高级管理层不想草率地作决定。公司高级管理层在实施优化组合方案之前还想知道可能要被解雇的员工的经理的工作地点，以便向他们的经理了解更详细的情况。例 7-12 的查询语句就可以用来完成这一要求。

例 7-12

```
SQL> SELECT w.empno "W_Number",w.ename "W_Name",w.job "W_Job",w.sal "W_Salary"
          ,m.empno "M_Number", m.ename "M_Name", d.loc "Location"
     FROM emp w, manager m, dept d
     WHERE w.mgr = m.empno
     AND   m.deptno = d.deptno
     AND   w.job IN ('CLERK', 'ANALYST')
     ORDER BY w.job, w.sal;
```

例 7-12 结果

```
W_Number W_Name     W_Job      W_Salary  M_Number M_Name    Location
---------- --------- --------- ---------- ---------- ---------- --------
      7788  SCOTT       ANALYST        3000      7566  JONES       DALLAS
      7902  FORD        ANALYST        3000      7566  JONES       DALLAS
      7369  SMITH       CLERK           800      7902  FORD        DALLAS
      7900  JAMES       CLERK           950      7698  BLAKE       CHICAGO
      7876  ADAMS       CLERK          1100      7788  SCOTT       DALLAS
      7934  MILLER      CLERK          1300      7782  CLARK       NEW YORK
已选择 6 行。
```

例 7-12 查询的思路是这样的：

（1）在员工表（emp）中找到员工经理号 w.mgr。

（2）利用员工经理号 w.mgr 在经理表（manager）中找到经理的信息。

（3）再利用经理的部门号（deptno）在部门表（dept）找到经理所在部门的信息。

（4）把员工的职位（job）不是文员（CLERK）或分析员（ANALYST）的员工排斥在查询结果以外。

（5）把查询结果按员工的职位（job）和工资（sal）由小到大排序并显示。

7.15　不等连接

在以上介绍的连接中其连接运算符都为等号（=）。我们也可以使用其他的运算符，其他的运算符所产生的连接叫不等连接。在介绍不等连接之前，我们先介绍一个在不等连接例子中要用到的一个表（salgrade）。您可以使用 SQL*Plus 命令 DESC 来得到它的结构，如例 7-13。

例 7-13

```
SQL> DESC salgrade
```

例 7-13 结果

```
名称                                  是否为空? 类型
----------------------------------- -------- ------
GRADE                                         NUMBER
LOSAL                                         NUMBER
HISAL                                         NUMBER
```

您也可以使用例 7-14 的查询语句来得到该表中所存的数据。

例 7-14

```
SQL> SELECT * FROM salgrade;
```

例 7-14 结果

```
    GRADE      LOSAL      HISAL
-------- ---------- ----------
       1        700       1200
       2       1201       1400
       3       1401       2000
       4       2001       3000
       5       3001       9999
```

那么 salgrade 表在商业运作中到底有什么用处呢？大家知道公司招工时免不了要和一些应征者就工资的多少讨价还价。公司负责招工的人就可以用这个表所提供的信息作为讨价还价的基准点。例如，所招聘的工作定为 3 级，公司负责招工的人对于任何高于 2000 元以上的要求都要予以拒绝，同时，公司也会把该级新员工的工资定在 1401 元以上。

例 7-15 的查询将显示工资级别在 3~5 级之间的所有员工。

例 7-15

```
SQL> SELECT e.empno, e.ename, e.job, e.sal, s.grade
  2  FROM emp e, salgrade s
  3  WHERE e.sal BETWEEN s.losal AND s.hisal
  4  AND s.grade > 2;
```

例 7-15 结果

```
     EMPNO  ENAME       JOB               SAL        GRADE
---------- ---------- --------- ---------- --------
      7499  ALLEN       SALESMAN          1600            3
      7844  TURNER      SALESMAN          1500            3
      7566  JONES       MANAGER           2975            4
      7698  BLAKE       MANAGER           2850            4
      7782  CLARK       MANAGER           2450            4
      7788  SCOTT       ANALYST           3000            4
      7902  FORD        ANALYST           3000            4
      7839  KING        PRESIDENT         5000            5
已选择 8 行。
```

在例 7-15 的查询中我们使用了 BETWEEN AND 作为连接运算符，该运算符不是等号（=），所以这个连接称为不等连接。

7.16 外 连 接

为了介绍外连接，您可以先使用例 7-16 的查询语句来看看 dept 表中的信息。

例 7-16

```
SQL> SELECT *
  2  FROM dept;
```

例 7-16 结果

```
    DEPTNO  DNAME           LOC
---------- -------------- --------
        10  ACCOUNTING      NEW YORK
        20  RESEARCH        DALLAS
        30  SALES           CHICAGO
        40  OPERATIONS      BOSTON
```

从例 7-16 查询的结果中，您会发现第 40 号部门并没有显示在例 7-1 的相等连接的结果中。这是因为相等连接只显示与 deptno 相等的查询结果，而 emp 表中并没有 deptno 40，所以 40 号部门不会显示在结果中。有时我们需要显示像 deptno 40 这样的记录。

您可以使用例 7-17 所示的外连接来满足以上的需要。

例 7-17

```
SQL> SELECT empno, ename, sal, emp.deptno, dept.deptno, loc
  2  FROM emp, dept
  3  WHERE emp.deptno(+) = dept.deptno;
```

例 7-17 结果

```
      EMPNO ENAME             SAL     DEPTNO     DEPTNO LOC
---------- ---------- ---------- ---------- --------- -----
      7782 CLARK             2450         10         10 NEW YORK
      7839 KING              5000         10         10 NEW YORK
      7934 MILLER            1300         10         10 NEW YORK
      7369 SMITH              800         20         20 DALLAS
      7876 ADAMS             1100         20         20 DALLAS
      7902 FORD              3000         20         20 DALLAS
      7788 SCOTT             3000         20         20 DALLAS
      7566 JONES             2975         20         20 DALLAS
      7499 ALLEN             1600         30         30 CHICAGO
      7698 BLAKE             2850         30         30 CHICAGO
      7654 MARTIN            1250         30         30 CHICAGO
      7900 JAMES              950         30         30 CHICAGO
      7844 TURNER            1500         30         30 CHICAGO
      7521 WARD              1250         30         30 CHICAGO
                                                     40 BOSTON
已选择 15 行。
```

例 7-17 的显示结果多了一行。该行即为第 40 号部门，其地点在波斯顿（BOSTON）。该部门没有雇用任何员工（这可能是一个刚成立的新部门）。

外连接的连接运算符为（+），该连接运算符既可以放在等号的左面也可以放在等号的右面，但一定要放在缺少相应信息的那一面，如放在 emp.deptno 所在的一方。我们可以把例 7-17 改写成例 7-18 的查询语句。

例 7-18

```
SQL> SELECT empno, ename, sal, emp.deptno, dept.deptno, loc
   2  FROM emp, dept
3  WHERE dept.deptno = emp.deptno(+);
```

如果感兴趣的话，您可以在 SQL*Plus 下输入例 7-18 的查询语句。您会发现例 7-18 显示的结果与例 7-17 的一样。

7.17　SQL:1999 语法的连接

Oracle 9i 对多表连接的语法进行了扩充，它开始支持美国国家标准化组织（ANSI）的 SQL 标准 SQL:1999 的语法。不过 SQL:1999 语法连接语句并没有提供效率方面的好处。

7.18　SQL:1999 语法的自然连接

例 7-19 的查询语句是使用 SQL:1999 语法的一个自然连接的例子。

例 7-19

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno, d.loc
  2  FROM emp e
  3  CROSS JOIN dept d
  4  ORDER BY d.loc;
```

例 7-19 也是一个笛卡儿乘积（乘积连接），它所产生的结果与例 7-4 的完全相同。在例 7-19 中阴影部分为 SQL:1999 语法所使用的关键字。本章以下各节还将沿用这一约定。

7.19 使用USING子句的连接

例 7-20 的查询语句是一个使用 USING 子句的连接。

例 7-20

```
SQL> SELECT e.empno, e.ename, e.sal, deptno, d.loc
  2  FROM emp e
  3  JOIN dept d
  4  USING (deptno)
  5  ORDER BY d.loc;
```

例 7-20 结果

EMPNO	ENAME	SAL	DEPTNO	LOC
7499	ALLEN	1600	30	CHICAGO
7521	WARD	1250	30	CHICAGO
7654	MARTIN	1250	30	CHICAGO
7900	JAMES	950	30	CHICAGO
7844	TURNER	1500	30	CHICAGO
7698	BLAKE	2850	30	CHICAGO
7369	SMITH	800	20	DALLAS
7902	FORD	3000	20	DALLAS
7876	ADAMS	1100	20	DALLAS
7566	JONES	2975	20	DALLAS
7788	SCOTT	3000	20	DALLAS
7782	CLARK	2450	10	NEW YORK
7839	KING	5000	10	NEW YORK
7934	MILLER	1300	10	NEW YORK

已选择 14 行。

例 7-20 所显示的结果与例 7-1 的相等连接的结果完全相同。在使用 USING 子句的连接时要注意以下的约定：

- NATURAL JOIN 子句和 USING 子句是互斥的，即两个子句只能使用一个。

- 在所引用的列中不能使用表名或列名（如例 7-20 中的 deptno）。
- 当有多列匹配时，使用 USING 子句只匹配其中的一列。

7.20　使用ON子句的连接

例 7-21 的查询语句是一个使用 ON 子句的连接。

例 7-21

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno, d.loc
     FROM emp e
     JOIN dept d
     ON (e.deptno = d.deptno)
     ORDER BY d.loc;
```

例 7-21 结果

EMPNO	ENAME	SAL	DEPTNO	LOC
7499	ALLEN	1600	30	CHICAGO
7521	WARD	1250	30	CHICAGO
7654	MARTIN	1250	30	CHICAGO
7900	JAMES	950	30	CHICAGO
7844	TURNER	1500	30	CHICAGO
7698	BLAKE	2850	30	CHICAGO
7369	SMITH	800	20	DALLAS
7902	FORD	3000	20	DALLAS
7876	ADAMS	1100	20	DALLAS
7566	JONES	2975	20	DALLAS
7788	SCOTT	3000	20	DALLAS
7782	CLARK	2450	10	NEW YORK
7839	KING	5000	10	NEW YORK
7934	MILLER	1300	10	NEW YORK

已选择 14 行。

例 7-21 所显示的结果与例 7-1 的相等连接的结果完全相同。在使用 ON 子句的连接时要注意以下的约定：

- 在所引用的列中要使用表名或列名（如例 7-21 中的 e.deptno 和 d.deptno）。
- 相等连接条件放在 ON 子句中。

7.21　使用ON子句的多表连接和附加条件

例 7-22 的查询语句是一个使用 ON 子句的多表连接。

例 7-22

```
SQL> SELECT w.empno "W_Number",w.ename "W_Name",w.job "W_Job",w.sal "W_Salary"
        ,m.empno "M_Number", m.ename "M_Name", d.loc "Location"
FROM emp w
JOIN manager m
ON   w.mgr = m.empno
JOIN dept d
ON   m.deptno = d.deptno
WHERE w.job IN ('CLERK', 'ANALYST')
ORDER BY w.job, w.sal;
```

例 7-22 结果

W_Number	W_Name	W_Job	W_Salary	M_Number	M_Name	Location
7788	SCOTT	ANALYST	3000	7566	JONES	DALLAS
7902	FORD	ANALYST	3000	7566	JONES	DALLAS
7369	SMITH	CLERK	800	7902	FORD	DALLAS
7900	JAMES	CLERK	950	7698	BLAKE	CHICAGO
7876	ADAMS	CLERK	1100	7788	SCOTT	DALLAS
7934	MILLER	CLERK	1300	7782	CLARK	NEW YORK

已选择 6 行。

例 7-22 的查询语句是用 SQL:1999 语法的一个多表连接，它所显示的结果与例 7-12 的相等连接的结果完全相同。

例 7-22 的查询中包含了一个附加条件，该条件放在 WHERE 子句中。我们还可以直接在查询语句中用 AND 加入附加条件，如例 7-23 的查询语句。

例 7-23

```
SQL>SELECT w.empno "W_Number", w.ename "W_Name",w.job "W_Job" w.sal "W_Salary"
        ,m.empno "M_Number", m.ename "M_Name", d.loc "Location"
FROM emp w
JOIN manager m
ON   w.mgr = m.empno
JOIN dept d
ON   m.deptno = d.deptno
AND  w.job IN ('CLERK', 'ANALYST')
ORDER BY w.job, w.sal;
```

例 7-23 结果

W_Number	W_Name	W_Job	W_Salary	M_Number	M_Name	Location
7788	SCOTT	ANALYST	3000	7566	JONES	DALLAS
7902	FORD	ANALYST	3000	7566	JONES	DALLAS

```
      7369 SMITH      CLERK           800       7902 FORD       DALLAS
      7900 JAMES      CLERK           950       7698 BLAKE      CHICAGO
      7876 ADAMS      CLERK          1100       7788 SCOTT      DALLAS
      7934 MILLER     CLERK          1300       7782 CLARK      NEW YORK
已选择 6 行。
```

显然，例 7-23 显示的结果与例 7-22 完全一样，查询语句中只要把例 7-22 的关键字 WHERE 换成 AND 就行了。

以上我们讨论的连接属于内连接。下面我们将讨论外连接。

7.22　左外连接

例 7-24 的查询语句是一个左外连接的例子。

例 7-24

```
SQL> SELECT empno, ename, sal, emp.deptno, dept.deptno, loc
  2  FROM dept
  3  LEFT OUTER JOIN emp
  4  ON (emp.deptno = dept.deptno);
```

例 7-24 结果

```
     EMPNO ENAME             SAL     DEPTNO     DEPTNO LOC
---------- ---------- ---------- ---------- ---------- ---------
      7369 SMITH             800         20         20 DALLAS
      7499 ALLEN            1600         30         30 CHICAGO
      7521 WARD             1250         30         30 CHICAGO
      7566 JONES            2975         20         20 DALLAS
      7654 MARTIN           1250         30         30 CHICAGO
      7698 BLAKE            2850         30         30 CHICAGO
      7782 CLARK            2450         10         10 NEW YORK
      7788 SCOTT            3000         20         20 DALLAS
      7839 KING             5000         10         10 NEW YORK
      7844 TURNER           1500         30         30 CHICAGO
      7876 ADAMS            1100         20         20 DALLAS
      7900 JAMES             950         30         30 CHICAGO
      7902 FORD             3000         20         20 DALLAS
      7934 MILLER           1300         10         10 NEW YORK
                                                    40 BOSTON
已选择 15 行。
```

例 7-24 所显示的结果与例 7-17 的外连接的结果完全相同，只不过它使用的是 SQL:1999 语法而已。

7.23　右外连接

例 7-25 的查询语句是一个右外连接的例子。

例 7-25

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno, d.deptno, d.loc
   2  FROM emp e
   3  RIGHT OUTER JOIN dept d
   4  ON (e.deptno = d.deptno);
```

例 7-25 结果

EMPNO	ENAME	SAL	DEPTNO	DEPTNO	LOC
7369	SMITH	800	20	20	DALLAS
7499	ALLEN	1600	30	30	CHICAGO
7521	WARD	1250	30	30	CHICAGO
7566	JONES	2975	20	20	DALLAS
7654	MARTIN	1250	30	30	CHICAGO
7698	BLAKE	2850	30	30	CHICAGO
7782	CLARK	2450	10	10	NEW YORK
7788	SCOTT	3000	20	20	DALLAS
7839	KING	5000	10	10	NEW YORK
7844	TURNER	1500	30	30	CHICAGO
7876	ADAMS	1100	20	20	DALLAS
7900	JAMES	950	30	30	CHICAGO
7902	FORD	3000	20	20	DALLAS
7934	MILLER	1300	10	10	NEW YORK
				40	BOSTON

已选择 15 行。

很显然，例 7-25 的结果与例 7-17 的结果也是完全相同的，只是写法不同而已。实际上例 7-25 相当于例 7-26 的查询语句。

例 7-26

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno, d.deptno, d.loc
   2  FROM emp e, dept d
3  WHERE d.deptno = e.deptno(+);
```

7.24　全外连接

在介绍全外连接之前，我们先做些准备工作。还记得我们在例 7-10 中所创建的表

manager 吗？现在我们将所有的第 30 号部门的员工的部门号置为空（NULL），例 7-27 的 SQL 语句就用来完成这一任务（读者只需要按照例 7-27 和例 7-28 的 SQL 语句原样输入即可，该语句将在数据操作语言——DML 中详细介绍）。

例 7-27

```
SQL> update manager
  2  set deptno = NULL
  3  WHERE deptno = 30;
```

例 7-27 结果

```
已更新 6 行。
```

例 7-28

```
SQL> commit;
```

例 7-28 结果

```
提交完成。
```

例 7-29 的查询语句是使用表 manager 和表 dept 的一个左外连接的例子。

例 7-29

```
SQL> SELECT empno, ename, sal, manager.deptno, dept.deptno, loc
  2  FROM manager
  3  LEFT OUTER JOIN dept
  4  ON (manager.deptno = dept.deptno);
```

例 7-29 结果

```
     EMPNO ENAME             SAL     DEPTNO      DEPTNO LOC
---------- ---------- ---------- ------ ----------- --------
      7934 MILLER           1300         10          10 NEW YORK
      7839 KING             5000         10          10 NEW YORK
      7782 CLARK            2450         10          10 NEW YORK
      7902 FORD             3000         20          20 DALLAS
      7876 ADAMS            1100         20          20 DALLAS
      7788 SCOTT            3000         20          20 DALLAS
      7566 JONES            2975         20          20 DALLAS
      7369 SMITH             800         20          20 DALLAS
      7900 JAMES             950
      7844 TURNER           1500
      7698 BLAKE            2850
      7654 MARTIN           1250
      7521 WARD             1250
      7499 ALLEN            1600
已选择 14 行。
```

因为例 7-29 是一个左外连接查询，而 manager 表在左面，所以该查询的结果为 manager

表中的所有记录，包括该表中那些在 dept 表中没有相匹配的 deptno 列的记录。其实，该查询等价于例 7-30 的查询语句。

例 7-30

```
SQL> SELECT empno, ename, sal, manager.deptno, dept.deptno, loc
  2  FROM manager, dept
  3  WHERE dept.deptno(+) = manager.deptno;
```

例 7-31 的查询语句是使用表 manager 和表 dept 的一个右外连接的例子。

例 7-31

```
SQL> SELECT empno, ename, sal, manager.deptno, dept.deptno, loc
  2  FROM manager
  3  RIGHT OUTER JOIN dept
  4  ON (manager.deptno = dept.deptno);
```

例 7-31 结果

EMPNO	ENAME	SAL	DEPTNO	DEPTNO	LOC
----------	----------	--------	----------	--------	--------
7369	SMITH	800	20	20	DALLAS
7566	JONES	2975	20	20	DALLAS
7782	CLARK	2450	10	10	NEW YORK
7788	SCOTT	3000	20	20	DALLAS
7839	KING	5000	10	10	NEW YORK
7876	ADAMS	1100	20	20	DALLAS
7902	FORD	3000	20	20	DALLAS
7934	MILLER	1300	10	10	NEW YORK
				30	CHICAGO
				40	BOSTON

已选择 10 行。

因为例 7-31 的查询语句是一个右外连接查询，manager 表在左面，而 dept 表在右面，所以该查询的结果为 dept 表中的所有记录，包括该表中那些在 manager 表中没有相匹配的 deptno 列的记录。其实，该查询等价于例 7-32 的查询语句。

例 7-32

```
SQL> SELECT empno, ename, sal, manager.deptno, dept.deptno, loc
  2  FROM manager, dept
  3  WHERE dept.deptno = manager.deptno(+);
```

例 7-33 的查询语句是使用表 manager 和表 dept 的一个全外连接的例子。

例 7-33

```
SQL> SELECT empno, ename, sal, manager.deptno, dept.deptno, loc
  2  FROM manager
```

```
3  FULL OUTER JOIN dept
4  ON (manager.deptno = dept.deptno);
```

例 7-33 结果

```
  EMPNO ENAME              SAL     DEPTNO     DEPTNO LOC
---------- ---------- -------- ---------- -------- --------
      7934 MILLER         1300         10         10 NEW YORK
      7839 KING           5000         10         10 NEW YORK
      7782 CLARK          2450         10         10 NEW YORK
      7902 FORD           3000         20         20 DALLAS
      7876 ADAMS          1100         20         20 DALLAS
      7788 SCOTT          3000         20         20 DALLAS
      7566 JONES          2975         20         20 DALLAS
      7369 SMITH           800         20         20 DALLAS
      7900 JAMES           950
      7844 TURNER         1500
      7698 BLAKE          2850
      7654 MARTIN         1250
      7521 WARD           1250
      7499 ALLEN          1600
                                                30 CHICAGO
                                                40 BOSTON
已选择 16 行。
```

例 7-33 的查询语句显示的结果既包括了 manager 表中的所有记录，也包括了 dept 表中的所有记录。

如果读者对 SQL:1999 语法的连接感到困惑的话，请不要担心，它们与 Oracle 的多表连接产生的结果完全相同，只是表示的方法不一样而已。因此，即使读者没有掌握 SQL:1999 语法的连接也不会影响书写 SQL 语句和以后的学习。等读者熟练掌握了 Oracle 的多表连接方法之后就会发现 SQL:1999 语法其实也不难学习。

7.25　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 为什么引入数据库的规范化（Normalization）？
- 主键（Primary Key）和实体完整性（Entity Integrity）。
- 重复组（Repeating Groups）或重复属性。
- 第一范式（1NF）。
- 为什么引入控制冗余？
- 外键（Foreign Key）和引用完整性（Referential Integrity）。

- 依赖关系图。
- 什么是部分依赖关系？
- 第二范式（2NF）。
- 什么是传递依赖关系？
- 第三范式（2NF）。
- 相等连接（Equijoin）。
- 自连接（Self join）。
- 不等连接（Non-equijoin）。
- 外连接（Outer join）。
- 两个以上的表的连接。
- 连接中表别名的使用。
- 如何避免笛卡儿乘积？
- SQL:1999 语法的一个自然连接。
- SQL:1999 语法的使用 USING 子句的连接。
- SQL:1999 语法的使用 ON 子句的连接。
- SQL:1999 语法的左外连接。
- SQL:1999 语法的右外连接。
- SQL:1999 语法的全外连接。

第8章

子查询

假设一个偶然的机会，您公司的老总发现一个叫 SMITH 的普通员工可以做不少高级职员的工作。老总很自然地推断其他与 SMITH 职位（job）相同的员工也许能做不少高级职员的工作。老总知道 SMITH 的工资是最低的，因为相同职位的员工工资应相差不多，所以用这些人来做高级职员的工作可以为公司节约不少钱。以下就是如何构造查询语句来帮助他得到他想要的信息。

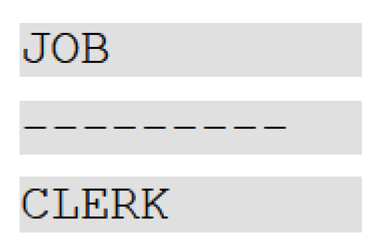
8.1 为什么引入单行子查询

现在老总让您帮助他找到职位与 SMITH 相同的所有员工。当然，您可以用例 8-1 和例 8-2 的查询语句来完成老总的重托。

例 8-1

```
SQL> SELECT job
  2  FROM   emp
  3  WHERE  ename = 'SMITH';
```

例 8-1 结果



```
JOB
---------
CLERK
```

从例 8-1 的查询结果得到 SMITH 的职位为文员（CLERK），您再把 CLERK 放在例 8-2 查询语句的 WHERE 子句中。

例 8-2

```
SQL> SELECT empno, ename, sal, job
  2  FROM emp
  3  WHERE job = 'CLERK';
```

例 8-2 结果

```
     EMPNO ENAME             SAL JOB
---------- ---------- ---------- ---------
      7369 SMITH             800 CLERK
      7876 ADAMS            1100 CLERK
```

```
      7900  JAMES               950  CLERK
      7934  MILLER             1300  CLERK
```

虽然例 8-2 的结果正是您的老总所需要的，但这一方法过于繁琐。另外，如果您现在正在开发一个软件，希望把例 8-1 和例 8-2 的查询语句嵌在该软件中，您可能会发现这是非常困难的，这也许就是引入子查询的原因之一。

8.2 WHERE子句中的单行子查询

您可以把例 8-1 和例 8-2 的查询语句集成在一个主查询（Main query）和一个子查询（Subquery）中，如例 8-3 的查询语句。

例 8-3

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  job =
  4               (SELECT job
  5                FROM   emp
  6                WHERE ename = 'SMITH');
```

例 8-3 结果

```
     EMPNO  ENAME               SAL  JOB
----------  ----------  ----------  ---------
      7369  SMITH               800  CLERK
      7876  ADAMS              1100  CLERK
      7900  JAMES               950  CLERK
      7934  MILLER             1300  CLERK
```

例 8-3 的查询语句包含了一个子查询，它与例 8-2 显示的结果完全相同。

例 8-3 中括号内的查询叫子查询（Subquery）或内查询（Inner query），括号外的查询叫主查询（Main query）或外查询（Outer query）。

Oracle 系统执行该查询语句的顺序如下：

（1）Oracle 首先执行括号中的子查询。

（2）返回 SMITH 的职位（job）为文员（CLERK）。

（3）主查询（Main query）把 CLERK 放在 WHERE 子句中之后执行主查询。

单行子查询的比较关系符包括>（大于）、>=（大于等于）、<（小于）、<=（小于等于）、=（等于）和<>或！=（不等于）。

单行子查询除了可以放在 WHERE 子句中，还可以放在 HAVING 子句和 FROM 子句中。

在书写单行子查询时要注意以下要求：

- 单行子查询使用单行比较关系符。

- 单行子查询放在单行比较关系符的右边。
- 单行子查询放在括号中。
- 单行子查询中不能使用 ORDER BY 子句。

根据以上的要求，单行子查询必须返回单一的行。因此，例 8-3 并不是一个好的子查询例子，因为在一个大型公司中员工的名字很难唯一。也许最稳妥的办法是在子查询的条件中使用主键（Primary Key）。

如果现在老总让您帮助他查找职位与 SMITH 相同，而工资不超过 ADAMS 的所有员工，您又该如何写这一查询语句呢？您可以使用例 8-4 的查询语句来完成老总的要求。

例 8-4

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  job =
  4              (SELECT job
  5               FROM   emp
  6               WHERE ename = 'SMITH')
  7  AND    sal <=
  8              (SELECT sal
  9               FROM   emp
 10               WHERE  ename = 'ADAMS');
```

例 8-4 结果

EMPNO	ENAME	SAL	JOB
7369	SMITH	800	CLERK
7876	ADAMS	1100	CLERK
7900	JAMES	950	CLERK

从例 8-4 结果可以看出，ADAMS 的工资为 1100，所显示的员工都是文员（CLERK），而且他们的工资都不高于 ADAMS。这正是您的老总所需要的信息。

您可以在一个查询语句中使用多个子查询，这些子查询所构成的条件再与逻辑运算符连接在一起。主查询和子查询的数据可以来自不同的表，而且每个子查询的数据也可以来自不同的表。

如果您的老总现在想知道哪些员工的工资在平均线以下，以便把更多的工作交给他们来做，您可以用例 8-5 的包含了子查询的 SQL 语句来完成这一重任。

例 8-5

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  sal <
  4              (SELECT AVG(sal)
  5               FROM   emp);
```

例 8-5 结果

```
     EMPNO ENAME                   SAL JOB
---------- ---------- ---------- ---------
      7369 SMITH                   800 CLERK
      7499 ALLEN                  1600 SALESMAN
      7521 WARD                   1250 SALESMAN
      7654 MARTIN                 1250 SALESMAN
      7844 TURNER                 1500 SALESMAN
      7876 ADAMS                  1100 CLERK
      7900 JAMES                   950 CLERK
      7934 MILLER                 1300 CLERK
已选择 8 行。
```

在例 8-5 的子查询中一定不能含有 GROUP BY 子句，因为该子句将使子查询返回多值。如果在子查询中加入了 GROUP BY 子句，Oracle 会返回错误信息，如例 8-6 所示。

例 8-6

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  sal <
  4              (SELECT AVG(sal)
  5               FROM   emp
  6               GROUP BY job);
```

例 8-6 结果

```
(SELECT AVG(sal)
*
ERROR 位于第 4 行:
ORA-01427: 单行子查询返回多于一个行
```

如果子查询中的 WHERE 子句的条件不成立，查询显示的结果又该是什么？请看例 8-7 的查询语句。

例 8-7

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  ename =
  4               (SELECT ename
  5                FROM   emp
  6                WHERE  ename = 'ADAM');
```

例 8-7 结果

```
未选定行。
```

在例 8-7 的查询语句中我们将 ADAMS 错误地写成了 ADAM，而在 emp 表中没有 ename

为 ADAM 的记录，所以子查询返回零行。这相当于主查询中的 WHERE 子句变成了"WHERE ename = NULL"，我们知道空值不等于任何值，所以主查询返回的结果为"未选定行"。因此，写查询语句时一定要保证子查询中的 WHERE 子句的条件成立并返回唯一的值。

我们已经介绍了 WHERE 子句中的单行子查询。单行子查询除了可以放在 WHERE 子句中外，还可以放在 HAVING 子句中。

8.3　HAVING子句中的单行子查询

如果您的老总让您打印一份平均工资高于最低平均工资（按职位分类）的所有职位的名单。您可以使用例 8-8 所示的查询语句来完成他的要求（其实，您在 SELECT 子句中只包含 job 和 AVG(sal)就可以了。为了给老总更多的信息，也包含 MIN(sal)和 MAX(sal)）。

例 8-8

```
SQL> SELECT job, MIN(sal), AVG(sal), MAX(sal)
     FROM emp
     WHERE job NOT LIKE 'PRESID%'
     GROUP BY job
     HAVING AVG(sal) > (
                        SELECT MIN(AVG(sal))
                        FROM emp
                        GROUP BY job);
```

例 8-8 结果

JOB	MIN(SAL)	AVG(SAL)	MAX(SAL)
ANALYST	3000	3000	3000
MANAGER	2450	2758.33333	2975
SALESMAN	1250	1400	1600

看到您所打印的清单，您的老总要好好地研究一下，以决定把哪些工作移交给文员来做，因为文员的平均工资最低。之后公司可解雇多余的高薪员工以达到降低成本（优化组合）的目的。

8.4　FROM子句中的单行子查询

老总对您的工作很满意，不过他还想知道除了文员以外，所有工资高于所任职位平均工资的员工。您可以使用例 8-9 的查询语句来完成这一工作。

例 8-9

```
SQL> SELECT e.empno, e.ename, e.sal, e.job, a.avesal
     FROM   emp e, (SELECT job, AVG(sal) avesal
```

```
               FROM   emp
               GROUP BY job) a
WHERE e.job = a.job
AND   e.sal > a.avesal
AND   e.job != 'CLERK';
```

例 8-9 结果

EMPNO	ENAME	SAL	JOB	AVESAL
7566	JONES	2975	MANAGER	2758.33333
7698	BLAKE	2850	MANAGER	2758.33333
7499	ALLEN	1600	SALESMAN	1400
7844	TURNER	1500	SALESMAN	1400

显然例 8-9 显示的结果要比例 8-8 的更具体，也许对公司的优化组合的重大决策更有用。

8.5 多行子查询

在前面几节中我们介绍了单行子查询。现在我们介绍另一类子查询，即多行子查询。多行子查询使用多行比较操作符，它返回多行。

多行比较操作符包括 IN、ANY 和 ALL。

我们在以后几节中将通过例子来分别介绍由这几个比较操作符组成的多行子查询。

1. 使用 IN 操作符的多行子查询

现在您的老总想知道除了文员和他自己以外，哪些人的工资为所任职位最高的。这些人也许是他最不喜欢的人，即最想解雇的人选。您可以使用例 8-10 的查询语句来完成他的这一要求。

例 8-10

```
SQL> SELECT empno, ename, job, sal
  FROM   emp
  WHERE  sal IN (SELECT MAX(sal)
               FROM   emp
               GROUP BY job)
  AND job <> 'CLERK'
  AND job NOT LIKE 'PRES%';
```

例 8-10 结果

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	1600
7566	JONES	MANAGER	2975

```
     7788 SCOTT      ANALYST         3000
     7902 FORD       ANALYST         3000
```

比较操作符 IN 的含义为子查询返回列表中的任何一个。IN 操作符比较子查询返回列表中的每一个值，并且显示任何相等的数据行。其实，例 8-10 可以由例 8-11 和例 8-12 的查询语句组成。

提示：

细心的读者可能会发现例 8-10 有点问题。如果 scott 的工资（sal）为 1600，他的信息也会显示在结果中，这与题目的要求是有出入的。这个问题可以通过 8.11 节的多列子查询来解决，如例 8-26 所示。随书光盘中有一个叫 ch8 的文本文件，其中是一个完整的演示例 8-10 问题和解决方案的例子，感兴趣的读者可以自己在电脑上运行一下。

例 8-11

```
SQL> SELECT MAX(sal)
  2  FROM   emp
  3  GROUP BY job;
```

例 8-11 结果

```
  MAX(SAL)
----------
      3000
      1300
      2975
      5000
      1600
```

例 8-12

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal IN (3000, 1300, 2975, 5000, 1600)
  4  AND job <> 'CLERK'
  5  AND job NOT LIKE 'PRES%';
```

例 8-12 结果

```
    EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
     7499 ALLEN      SALESMAN        1600
     7566 JONES      MANAGER         2975
     7788 SCOTT      ANALYST         3000
     7902 FORD       ANALYST         3000
```

您还可以将例 8-10 的查询语句改写成例 8-13 的 SQL 语句，您会发现例 8-13 返回的结果与例 8-10 完全一样。

例 8-13

```
SQL> SELECT empno, ename, job, sal
     FROM   emp
     WHERE  sal IN (SELECT MAX(sal)
                    FROM   emp
                    WHERE job <> 'CLERK'
                    AND job NOT LIKE 'PRES%'
                    GROUP BY job);
```

例 8-13 结果

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	1600
7566	JONES	MANAGER	2975
7788	SCOTT	ANALYST	3000
7902	FORD	ANALYST	3000

2．使用 ALL 操作符的多行子查询

假如您的老总想保留低收入的职员，他想知道哪些员工的工资比任何职位（job）的平均工资还低，这些员工也许是公司最喜欢的。您可以使用例 8-14 的查询语句来完成这一工作。

例 8-14

```
SQL> SELECT empno, ename, sal, job
     FROM   emp
     WHERE  sal < ALL
                    (SELECT AVG(sal)
                     FROM   emp
                     GROUP BY job);
```

例 8-14 结果

EMPNO	ENAME	SAL	JOB
7369	SMITH	800	CLERK
7900	JAMES	950	CLERK

ALL 操作符比较子查询返回列表中的每一个值。< ALL 为小于最小的；> ALL 为大于最大的；那么= ALL 是什么？请看例 8-15 的查询语句。

例 8-15

```
SQL> SELECT empno, ename, sal, job
     FROM   emp
     WHERE  sal = ALL
```

```
  4                (SELECT AVG(sal)
  5                 FROM   emp
  6                 GROUP BY job);
```

例 8-15 结果

```
未选定行
```

因为等于子查询返回列表中的所有值是不符合逻辑的，所以 Oracle 返回“未选定行”。

3．使用 ANY 操作符的多行子查询

如果您的老总想知道哪些员工的工资比最低的平均工资高（按职位），这些员工也许是公司不太喜欢的。您可以使用例 8-16 的查询语句来完成这一工作。

例 8-16

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  sal > ANY
  4                (SELECT AVG(sal)
  5                 FROM   emp
  6                 GROUP BY job);
```

例 8-16 结果

```
     EMPNO ENAME             SAL JOB
---------- ---------- ---------- ---------
      7499 ALLEN            1600 SALESMAN
      7521 WARD             1250 SALESMAN
      7566 JONES            2975 MANAGER
      7654 MARTIN           1250 SALESMAN
      7698 BLAKE            2850 MANAGER
      7782 CLARK            2450 MANAGER
      7788 SCOTT            3000 ANALYST
      7839 KING             5000 PRESIDENT
      7844 TURNER           1500 SALESMAN
      7876 ADAMS            1100 CLERK
      7902 FORD             3000 ANALYST
      7934 MILLER           1300 CLERK
已选择 12 行。
```

ANY 操作符比较子查询返回列表中的每一个值。< ANY 为小于最大的；> ANY 为大于最小的；那么= ANY 是什么？请看例 8-17 的查询语句。

例 8-17

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  sal = ANY
```

```
  4                (SELECT AVG(sal)
  5                 FROM   emp
  6                 GROUP BY job);
```

例 8-17 结果

```
     EMPNO ENAME                  SAL JOB
---------- ---------- ---------- ---------
      7788 SCOTT                3000 ANALYST
      7902 FORD                 3000 ANALYST
      7839 KING                 5000 PRESIDENT
```

= ANY 等价于 IN，我们可以使用例 8-18 的查询语句来验证这一点。

例 8-18

```
SQL> SELECT empno, ename, sal, job
  2  FROM   emp
  3  WHERE  sal IN
  4                (SELECT AVG(sal)
  5                 FROM   emp
  6                 GROUP BY job);
```

例 8-18 结果

```
     EMPNO ENAME                  SAL JOB
---------- ---------- ---------- ---------
      7788 SCOTT                3000 ANALYST
      7902 FORD                 3000 ANALYST
      7839 KING                 5000 PRESIDENT
```

8.6　子查询中的空值问题

假如您想知道哪些员工没有经理管，这些员工一般是公司高级管理层的成员。您可以试着使用例 8-19 的查询语句来完成这一工作。

例 8-19

```
SQL> SELECT e.empno, e.ename, e.sal, e.job
  2  FROM   emp e
  3  WHERE  e.mgr IN
  4                 (SELECT w.mgr
  5                  FROM   emp w
  6                  WHERE  w.mgr IS NULL);
```

例 8-19 结果

```
未选定行
```

例 8-19 的结果真是令人失望。这是因为子查询返回列表中包含了空值（NULL），而任何值都不等于空值，所以例 8-19 的查询是得不到任何结果的。

现在我们换一种方法，找到那些不在子查询列表中的员工，即有经理管的员工。我们重新改写例 8-19 的查询语句，如例 8-20 所示。

例 8-20

```
SQL> SELECT e.empno, e.ename, e.sal, e.job
  2  FROM   emp e
  3  WHERE  e.mgr NOT IN
  4                  (SELECT w.mgr
  5                   FROM   emp w
  6                   WHERE  w.mgr IS NULL);
```

例 8-20 结果

```
未选定行。
```

例 8-20 的结果同样令人失望。这是因为 NOT IN 等价于<>ALL，而子查询返回列表中包含了空值。我们知道任何值也都不能为不等于空值，所以例 8-20 的查询也得不到任何结果。

如果我们将例 8-20 查询语句的第 6 行，即子查询的 WHERE 子句改写成 WHERE w.mgr IS NOT NULL，其结果是不是我们所希望的那些没有经理管的员工呢？请看例 8-21 的查询语句。

例 8-21

```
SQL> SELECT e.empno, e.ename, e.sal, e.job
  2  FROM   emp e
  3  WHERE  e.mgr NOT IN
  4                  (SELECT w.mgr
  5                   FROM   emp w
  6                   WHERE  w.mgr IS NOT NULL);
```

例 8-21 结果

```
未选定行。
```

在例 8-21 的子查询中我们得到了所有非空的经理号，一个自然而逻辑的想法就是经理号不在这个非空的经理号列表中的记录就是我们要找的那些没有经理管的员工。这个想法本身并没有任何错误，造成例 8-21 得不到结果的原因又是空值，只不过这次空值是在表达式的左面而不是右面。

如果我们将例 8-21 的第 3 行，即主查询的 WHERE 子句改写成“WHERE e.mgr IN”，其结果为全部有经理管的员工。请看例 8-22 的查询语句。

例 8-22

```
SQL> SELECT e.empno, e.ename, e.sal, e.job
  2  FROM   emp e
```

```
  3  WHERE  e.mgr IN
  4                 (SELECT w.mgr
  5                  FROM   emp w
  6                  WHERE  w.mgr IS NOT NULL);
```

例 8-22 结果

```
    EMPNO ENAME             SAL JOB
--------- ---------- ---------- -------
     7788 SCOTT            3000 ANALYST
     7902 FORD             3000 ANALYST
     7499 ALLEN            1600 SALESMAN
     7521 WARD             1250 SALESMAN
     7900 JAMES             950 CLERK
     7844 TURNER           1500 SALESMAN
     7654 MARTIN           1250 SALESMAN
     7934 MILLER           1300 CLERK
     7876 ADAMS            1100 CLERK
     7566 JONES            2975 MANAGER
     7782 CLARK            2450 MANAGER
     7698 BLAKE            2850 MANAGER
     7369 SMITH             800 CLERK
已选择 13 行。
```

这次我们总算得到了结果，只不过是一个间接的结果。

上面的讨论再一次表明，在写 SQL 语句时，空值常常是一个令人头痛的问题。特别是在把 SQL 语句嵌在程序中而且操作的表又很大时，如果空值没有被妥善处理的话，可能会产生一些意想不到的结果，而且这些结果是随机的。

8.7 多列子查询

与以前的子查询不同，多列子查询要返回多列。多列子查询又分为成对比较子查询（Pairwise Comparison）和非成对比较（Nonpairwire Comparison）子查询。下面我们先来介绍成对比较的多列子查询。

1．成对比较的多列子查询

为了解释上的方便，在本节和 8.12 节，我们将使用以前创建的一个表 manager，并使用例 8-23、例 8-24 和例 8-25 的 SQL 语句对此表进行修改。

例 8-23

```
SQL> UPDATE manager
  2  SET    sal = 1300
  3  WHERE  empno = 7521;
```

例 8-23 结果

```
已更新 1 行。
```

例 8-24

```
SQL> UPDATE manager
  2  SET    sal = 1600
  3  WHERE  empno = 7782;
```

例 8-24 结果

```
已更新 1 行。
```

例 8-25

```
SQL> commit;
```

例 8-25 结果

```
提交完成。
```

注意：

在现阶段读者只需要按照例 8-23~例 8-25 原样输入即可，这些命令我们将在以后的章节中有详细介绍。

如果您想知道哪些职员是工资为所任职位最高的，也可以使用例 8-26 的查询语句来完成这一工作。

例 8-26

```
SQL> SELECT empno, ename, sal, job
  2  FROM   manager
  3  WHERE (sal, job) IN
  4                (SELECT MAX(sal), job
  5                 FROM   manager
  6                 GROUP  BY job);
```

例 8-26 结果

```
     EMPNO ENAME             SAL JOB
---------- ---------- ---------- -----
      7934 MILLER           1300 CLERK
      7499 ALLEN            1600 SALESMAN
      7566 JONES            2975 MANAGER
      7788 SCOTT            3000 ANALYST
      7902 FORD             3000 ANALYST
      7839 KING             5000 PRESIDENT
已选择 6 行。
```

在例 8-26 中子查询返回每一种职位的最高工资和职位的名称。之后主查询的每一行中的工资和职位都要与子查询返回列表中的最高工资和职位相比较，只有当两者同时完全匹配时才显示该数据行，这也就是所谓的成对比较的多列子查询。

2．非成对比较的多列子查询

如果您的老总想要知道哪些职员的工资与某一职位的最高工资相同，您可以使用例 8-27 的查询语句来完成他的要求。

例 8-27

```
SQL> SELECT empno, ename, sal, job
     FROM   manager
     WHERE  sal IN (SELECT MAX(sal)
                    FROM   manager
                    GROUP  BY job)
     AND    job IN (SELECT DISTINCT job
                    FROM manager);
```

例 8-27 结果

```
     EMPNO ENAME             SAL JOB
---------- ---------- ------ ---------
      7788 SCOTT            3000 ANALYST
      7902 FORD             3000 ANALYST
      7934 MILLER           1300 CLERK
      7782 CLARK            1600 MANAGER
      7566 JONES            2975 MANAGER
      7839 KING             5000 PRESIDENT
      7521 WARD             1300 SALESMAN
      7499 ALLEN            1600 SALESMAN
已选择 8 行。
```

例 8-27 的结果多了两行，一行是工资为 1600 的经理（1600 为推销员的最高工资），另一行是工资为 1300 的推销员（1300 为文员的最高工资）。例 8-27 就是所谓的非成对比较的多列子查询。

从例 8-26 和例 8-27 的结果可以知道，非成对比较的多列子查询的条件要宽松些，因为它并不要求把主查询的工资和职位与子查询返回列表中的最高工资和职位进行比较和两者同时完全匹配，只要主查询工资和职位在子查询返回列表中出现过就行。

8.8 小　　结

子查询就是嵌在某一查询子句中的查询。子查询分为单行子查询、多行子查询和多列子查询。单行子查询返回一行数据，单行比较关系符包括“>”、“>=”、“<”、“<=”、“=”和“<>”；多行子查询返回多行数据，多行比较关系符包括 IN、ALL 和 ANY。

当查询是基于未知的条件时，子查询很有用。子查询可以放在 WHERE、HAVING 和 FROM 子句中。子查询中还可以包含分组函数和 GROUP BY 子句，但不能包含 ORDER BY

子句。最后要注意，在包含子查询的查询语句中，如果空值处理不当也可能产生意想不到的结果。

8.9　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 引入单行子查询的原因。
- 什么是主查询（Main query）？
- 什么是子查询（Subquery）？
- 主查询（Main query）和子查询（Subquery）之间的关系。
- 单行子查询在 WHERE 子句中的使用。
- 单行比较关系符。
- 单行子查询在 HAVING 子句中的使用。
- 单行子查询在 FROM 子句中的使用。
- 多行子查询。
- 多行比较关系符。
- 如何处理子查询中的空值（NULL）问题?
- 子查询中的分组函数和 GROUP BY 子句。
- 子查询与 ORDER BY 子句。
- 多列子查询。

第9章

控制 SQL*Plus 的环境和数据字典简介

虽然在以前的各章中我们使用 SQL*Plus 时并未设置任何东西，而且查询操作和SQL*Plus 命令的显示结果也基本上能满足我们的要求，但如果您在工作中要产生一个商业报表，就需要调整和控制 SQL*Plus 的环境以得到所需要的输出，特别是这些商业报表的读者为非计算机专业人员时就显得尤为重要。

9.1 控制SQL*Plus的环境

可以通过使用 SET 命令来设置 SQL*Plus 的环境变量，从而达到控制 SQL*Plus 环境的目的。

SET 命令的格式如下：

SET 环境变量 变量的值

可以通过使用 SHOW 命令来显示 SQL*Plus 环境变量的配置。

SHOW 命令的格式如下：

SHOW 环境变量 | ALL

9.2 SQL*Plus的环境变量ECHO

下面用一个例子来解释 SET 命令的用法。首先查看一下 SQL*Plus 的环境变量 ECHO 的设置，您可以输入例 9-1 的 SQL*Plus 命令。

例 9-1

```
SQL> show echo
```

例 9-1 结果

```
echo OFF
```

现在您可以在 SQL*Plus 中输入例 9-2 的查询语句。

例 9-2

```
SQL> SELECT * FROM dept;
```

例 9-2 结果

```
  DEPTNO  DNAME           LOC
---------- -------------- --------
      10  ACCOUNTING      NEW YORK
      20  RESEARCH        DALLAS
      30  SALES           CHICAGO
      40  OPERATIONS      BOSTON
```

之后您再把例 9-2 的查询语句存在一个脚本文件中，如例 9-3 所示存在 d:\Oracle\ming\echo_sample 文件中。

例 9-3

```
SQL> save "d:\ Oracle\ming\echo_sample"
```

例 9-3 结果

```
已创建文件 d:\ Oracle\ming\echo_sample.sql
```

注意：

d:\Oracle\ming 这一目录（文件夹）必须在输入例 9-3 的命令之前用操作系统命令建立。可以指定任何您感兴趣的目录和文件名。

现在可以运行脚本文件 d:\Oracle\ming\echo_sample，输入例 9-4 的 SQL*Plus 命令。

例 9-4

```
SQL> @d:\ Oracle\ming\echo_sample
```

例 9-4 结果

```
  DEPTNO  DNAME           LOC
---------- -------------- --------
      10  ACCOUNTING      NEW YORK
      20  RESEARCH        DALLAS
      30  SALES           CHICAGO
      40  OPERATIONS      BOSTON
```

从例 9-4 的结果可以看出，运行脚本文件 d:\Oracle\ming\echo_sample 后，Oracle 只返回了查询的结果，而没有包含查询的命令。

现在可以用 SET 命令把 SQL*Plus 的环境变量 ECHO 设置成 ON，如例 9-5 所示。

例 9-5

```
SQL> set echo on
```

再使用 SHOW 命令查看 SQL*Plus 的环境变量 ECHO 的设置，如例 9-6 所示。

例 9-6

```
SQL> SHOW ECHO
```

例 9-6 结果

```
echo ON
```

现在您再使用如例 9-7 的 SQL*Plus 命令来运行脚本文件 d:\Oracle\ming\echo_sample。

例 9-7

```
SQL> @d:\ Oracle\ming\echo_sample
```

例 9-7 结果

```
SQL> SELECT * FROM dept
  2  /

    DEPTNO DNAME          LOC
---------- -------------- ------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

从例 9-7 的结果可以看出，运行脚本文件 d:\Oracle\ming\echo_sample 后，Oracle 不但返回了查询的结果，而且还返回了查询的命令。现在您应该对 SQL*Plus 的环境变量 ECHO 有所了解了吧！

为了看到所有的 SET 变量值，可以使用 SHOW ALL 命令，如例 9-8 所示。

例 9-8

```
SQL> show all
```

例 9-8 结果

```
appinfo 为 OFF 并且已设置为"SQL*Plus"
arraysize 15
autocommit OFF
autoprint OFF
autorecovery OFF
autotrace OFF
blockterminator "." (hex 2e)
btitle OFF and 为下一条 SELECT 语句的前几个字符
…
```

介绍完了 SQL*Plus 的环境变量 ECHO 的使用之后，再来讨论另一个可能会经常使用的环境变量 FEEDBACK。

9.3 SQL*Plus的环境变量FEEDBACK

您可以首先用 SHOW 命令来显示一下变量 FEEDBACK 的当前设置，如例 9-9 所示。

例 9-9

```
SQL> SHOW FEEDBACK
```

例 9-9 结果

```
用于 6 或更多行的 FEEDBACK ON
```

SET FEEDBACK 的命令格式如下：

SET FEED[BACK]{6|n|OFF|ON}

当查询选择的数据行数大于 n 时，显示返回的数据行数。n 为自然数，6 为 Oracle 的默认值。

例 9-9 的结果告诉我们 n 为 6，即当查询选择了至少 6 条记录时，显示返回的记录数。但如果查询选择的记录数少于 6 条，就不显示返回的记录数。可以使用例 9-10 和例 9-11 的 SQL 语句来验证这一点。

例 9-10

```
SQL> SELECT *
  2  FROM dept;
```

例 9-10 结果

```
    DEPTNO DNAME          LOC
---------- -------------- --------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

例 9-10 结果返回的记录只为 4 行，小于 6 条记录，所以 Oracle 不显示返回的记录数。

例 9-11

```
SQL> SELECT ename, sal
  2  FROM emp
  3  WHERE sal > 1500;
```

例 9-11 结果

```
ENAME             SAL
---------- ----------
ALLEN            1600
JONES            2975
BLAKE            2850
CLARK            2450
SCOTT            3000
KING             5000
FORD             3000
已选择 7 行。
```

例 9-11 结果返回的记录为 7 行，大于 6 条记录，所以 Oracle 显示返回的记录数“已选择 7 行。”

9.4　SQL*Plus其他常用的环境变量

除了 ECHO 和 FEEDBACK 外，还有一些其他常用的环境变量，我们在这里作一个简单的介绍，后面将结合例子对它们作进一步的解释。

- SET HEA[DING] { ON |OFF }：决定在报告中是否显示列的标题。
- SET ARRAY[SIZE] { 20|n }：限制SQL*Plus每次从数据库中获取的行数，最多为 5000 行。
- SET LINE[SIZE] { 80|n }：设置每行的字符数。
- SET PAGE[SIZE] { 24|n }：设置每页的行数。
- SET LONG { 80|n }：设置显示LONG、CLOB和NCLOB值时最长的字节宽度，最大值为 2GB。

9.5　SQL*Plus的COLUMN格式化命令

为了产生用户友好的输出，SQL*Plus 提供了一些格式化命令，其中使用得最多的是 COLUMN 命令。

COLUMN 命令的格式如下：

COL[UMN] [{ 列名|别名 } [可选项]]

可选项的格式为：CLE[AR]|FOR[MAT] 格式化模式|HEA[DING] 正文|JUS[TIFY] {对齐选项 }|NUL[L] 正文|PRI[NT]|NOPRI[NT]|…

在解释这些可选项之前，我们使用几个例子来帮助读者理解 COLUMN 命令的用法。

例 9-12

```
SQL> select * from dept;
```

例 9-12 结果

```
  DEPTNO DNAME          LOC
---------- -------------- ----------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

从例 9-12 显示的结果来看，DEPTNO 的宽度似乎有点大了。可以使用如例 9-13 所示的 SQL*Plus 命令来把该列的宽度设置为 6 位数字，之后再重新输入与例 9-12 完全相同的查询语句，如例 9-14 所示。

例 9-13

```
SQL> col deptno for 999999
```

例 9-14

```
SQL> select * from dept;
```

例 9-14 结果

```
DEPTNO  DNAME          LOC
------- -------------- -------------
     10 ACCOUNTING      NEW YORK
     20 RESEARCH       DALLAS
     30 SALES         CHICAGO
     40 OPERATIONS      BOSTON
```

从例 9-14 显示的结果可以看出，DEPTNO 的宽度确实缩小了，仅为 6 位数字。因为 DEPTNO 列为数字型，所以例 9-13 的 FOR[MAT]之后要用 999999。这里的 999999 是格式化模式，每一个 9 表示一位数字（不显示 0），6 个 9 就表示 6 位数字。

例 9-14 显示的结果表明，LOC 的宽度也有点大。可以使用如例 9-15 所示的 SQL*Plus 命令来把该列的宽度设置为 9 个字符，之后再重新输入与例 9-12 完全相同的查询语句，如例 9-16 所示。

例 9-15

```
SQL> col loc for a9
```

例 9-16

```
SQL>  select * from dept;
```

例 9-16 结果

```
DEPTNO  DNAME          LOC
------- -------------- ---------
     10 ACCOUNTING      NEW YORK
     20 RESEARCH       DALLAS
     30 SALES         CHICAGO
     40 OPERATIONS      BOSTON
```

从例 9-16 的结果可以看出，LOC 的宽度确实缩小了，仅为 9 个字符的宽度。因为 LOC 列为字符型，所以例 9-16 的 FOR[MAT]之后要使用 A9。这里的 A9 也是格式化模式，A9 表示 9 个字符的宽度。

您可能已经注意到了例 9-16 显示的标题不太好理解，特别是 LOC。可以使用例 9-17 的 SQL*Plus 命令把标题 LOC 设置为 Location，之后再重新输入与例 9-16 完全相同的查询语句，如例 9-18 所示。

例 9-17

```
SQL> col loc HEADING 'Location' FOR A9
```

例 9-18

```
SQL> select * from dept;
```

例 9-18 结果

```
 DEPTNO DNAME          Location
------- -------------- ---------
     10 ACCOUNTING      NEW YORK
     20 RESEARCH       DALLAS
     30 SALES         CHICAGO
     40 OPERATIONS      BOSTON
```

从例 9-18 显示的结果可以看出，标题 LOC 已变成了 Location。HEA[DING]是用来设置列标题的。如果 HEA[DING]之后的正文中包含了竖线（|），该正文将以竖线为分界线，将竖线左、右的正文分别显示在不同的行上（竖线右边的在下一行上）。

如果您的数据库字符集为中文，也可以将某一列的标题设置成中文。例如，可以使用例 9-19 和例 9-20 的命令将标题 LOC 变成“地点”并显示 dept 表的内容。

例 9-19

```
SQL> col loc HEADING '地 点' for a9
```

例 9-20

```
SQL> select * from dept;
```

例 9-20 结果

```
   DEPTNO DNAME           地 点
---------- -------------- ---------
        10 ACCOUNTING      NEW YORK
        20 RESEARCH       DALLAS
        30 SALES         CHICAGO
        40 OPERATIONS      BOSTON
```

一般在开发商业系统时，考虑到易读性，在产生报表时要把由英文表示的表名和列名转换成用户本国的语言（如中文）。这时，例 9-19 和例 9-20 的命令就派上了用场。

您已经修改了 DEPTNO 和 LOC 的显示格式及 LOC 的标题。如果已经记不清它们的格式，可以使用例 9-21 和例 9-22 的 SQL*Plus 命令来得到所需的信息。

例 9-21

```
SQL> COL loc
```

例 9-21 结果

```
COLUMN   loc ON
HEADING  'Location'
FORMAT   A9
```

例 9-22

```
SQL> col deptno
```

例 9-22 结果

```
COLUMN   deptno ON
FORMAT   999999
```

从例 9-21 和例 9-22 显示的结果可以看出，一旦设置某一列的格式或标题，这些设置就一直保留在 SQL*Plus 中。如果要把某一列的属性重新置回默认值，应使用 CLEAR 选项。现在我们通过例 9-23 和例 9-24 来说明 CLEAR 选项的用法和含义。

例 9-23

```
SQL> col loc clear
```

现在重新输入与例 9-12 完全相同的查询语句，如例 9-24 所示。

例 9-24

```
SQL> select * from dept;
```

例 9-24 结果

```
 DEPTNO DNAME          LOC
------- -------------- -------------
     10 ACCOUNTING      NEW YORK
     20 RESEARCH        DALLAS
     30 SALES          CHICAGO
     40 OPERATIONS      BOSTON
```

例 9-24 的结果与例 9-12 的结果完全相同，这说明 COLUMN 命令的 CLEAR 选项是把某一列的属性重新置回默认值。还可以通过使用 SQL*Plus 的 COL loc 命令来获得相关的信息。

我们已经介绍了格式化模式中的 An 和 9。下面通过例子来说明另外几种格式化模式，先看例 9-25。

例 9-25

```
SQL> SELECT ename, job, sal
  2  FROM emp
  3  WHERE sal >= 1500;
```

例 9-25 结果

```
ENAME      JOB              SAL
---------- --------- ----------
ALLEN      SALESMAN        1600
JONES      MANAGER         2975
BLAKE      MANAGER         2850
CLARK      MANAGER         2450
SCOTT      ANALYST         3000
KING       PRESIDENT       5000
```

```
TURNER      SALESMAN        1500
FORD        ANALYST         3000
已选择 8 行。
```

从例 9-25 显示的结果很难看出工资（sal）的单位到底是什么，可以先使用例 9-26 的 SQL*Plus 命令来格式化 SAL 列。

例 9-26

```
SQL> col sal for $99,999.99
```

“$”为美元符号，“,”为千位符号，“.”为小数点，9 为不显示前导 0。

现在重新输入与例 9-23 完全相同的查询语句，即例 9-27。

例 9-27

```
SQL> SELECT ename, job, sal
   2  FROM emp
   3  WHERE sal >= 1500;
```

例 9-27 结果

```
ENAME      JOB               SAL
---------- --------- -----------
ALLEN      SALESMAN    $1,600.00
JONES      MANAGER     $2,975.00
BLAKE      MANAGER     $2,850.00
CLARK      MANAGER     $2,450.00
SCOTT      ANALYST     $3,000.00
KING       PRESIDENT   $5,000.00
TURNER     SALESMAN    $1,500.00
FORD       ANALYST     $3,000.00
已选择 8 行。
```

例 9-27 显示的结果与例 9-25 的完全相同，只是工资的显示更容易理解而已。

我们用例 9-28 和例 9-29 来说明 COLUMN 命令的另外一个格式化模式 0。

例 9-28

```
SQL> col sal for $009,999.99
```

例 9-29

```
SQL> SELECT ename, job, sal
   2  FROM emp
   3  WHERE sal >= 1500;
```

例 9-29 结果

```
ENAME      JOB                SAL
---------- --------- ------------
ALLEN      SALESMAN   $001,600.00
```

```
JONES        MANAGER     $002,975.00
BLAKE        MANAGER     $002,850.00
CLARK        MANAGER     $002,450.00
SCOTT        ANALYST     $003,000.00
KING         PRESIDENT   $005,000.00
TURNER       SALESMAN    $001,500.00
FORD         ANALYST     $003,000.00
已选择 8 行。
```

从例 9-29 显示的结果可以看出，COLUMN 命令的格式化模式 0 是强迫显示前导 0。

如果货币的单位不是美元，我们又如何显示这一货币单位呢？可以使用 COLUMN 命令的另外一个格式化模式 L，如例 9-30 和例 9-31 所示。

例 9-30

```
SQL> col sal for L99,999.99
```

例 9-31

```
SQL> SELECT ename, job, sal
   2  FROM emp
3  WHERE sal >= 1500;
```

例 9-31 结果

```
ENAME        JOB                         SAL
---------- --------- -------------------
ALLEN        SALESMAN            RMB1,600.00
JONES        MANAGER             RMB2,975.00
BLAKE        MANAGER             RMB2,850.00
CLARK        MANAGER             RMB2,450.00
SCOTT        ANALYST             RMB3,000.00
KING         PRESIDENT           RMB5,000.00
TURNER       SALESMAN            RMB1,500.00
FORD         ANALYST             RMB3,000.00
已选择 8 行。
```

从例 9-31 显示的结果可以看出，COLUMN 命令的格式化模式 L 是显示本地货币的单位。因为该数据库的字符集为中文，所以本地货币的单位是 RMB。

9.6　SQL*Plus的其他格式化命令

除了 COLUMN 命令之外，SQL*Plus 还提供了一些其他的常用格式化命令，它们包括 TTITLE（Top Title）、BTITLE（Bottom Title）和 BREAK，其命令格式如下。

- TTI[TLE] [正文 |OFF|ON]：设置每页顶部的头标。
- BTI[TLE] [正文 |OFF|ON]：设置每页底部的脚标。

- BREAK ON 列名 [| 别名] [SKIP n]：去掉重复的行，并在断开点跳过 n 行。
- CLEAR BREAK：清除所有的 BREAK 设置。

为了使 BREAK 有效地工作，需要在设置断点的列上使用 ORDER BY 子句。

下面我们通过一个比较完整的例子来更详细地介绍 SQL*Plus 的一些常用的格式化命令，如例 9-32 所示。

例 9-32

```
REM  *** This is an employees' salary report for senior management team ***
REM  *** Strictly Confidential ***

SET PAGESIZE 25
SET LINESIZE 80
SET FEEDBACK OFF
TTITLE '====== Sun_Moon IT Company ====== |===== Employee Salary Report ====='
BTITLE 'Strictly Confidential !!!'
BREAK ON deptno SKIP 2
COLUMN deptno HEADING 'Department|Number' JUSTIFY CENTER FORMAT 99999999999
COLUMN job    HEADING 'Job|Category'      FORMAT A15
COLUMN AVG(sal)  HEADING 'Average|Salary' FORMAT L99,999.00
COLUMN COUNT(sal)  HEADING 'Employee|Number' JUSTIFY CENTER  FORMAT 999
COLUMN SUM(sal)  HEADING 'Summary|Salary' FORMAT L99,999.00

SELECT deptno, job, AVG(sal), COUNT(sal), SUM(sal)
FROM emp
GROUP BY deptno, job
/
SET FEEDBACK ON
```

现在我们来逐一解释例 9-32 中的命令。

- 以 REM 开始的语句为注释，即在运行时 Oracle 并不执行该语句。注释语句只是一些帮助理解脚本文件内容的解释，可以在注释语句中放入任何您认为有用的句子。
- SET PAGESIZE 25：设置报告的显示长度为 25 行。
- SET LINESIZE 80：设置每行的显示宽度为 80 字符。
- SET FEEDBACK OFF：关闭反馈。
- TTITLE… ：设置报告的头标为“====== Sun_Moon IT Company =====
====== Employee Salary Report =====”。
- BTITLE… ：设置报告的脚标为“Strictly Confidential !!!”。
- BREAK ON deptno SKIP 2：设置当 deptno 变化时换行并跳过两行。
- COLUMN deptno…：设置 deptno 的题标为 Department 中间对齐并显示 11 位数字（Number）。
- COLUMN job…：设置 job 的题标为 Job 并显示 15 个字符（Category）。

- COLUMN AVG(sal)…：设置 AVG(sal)的题标为 Average 并显示 7 位数字，小数点后两位，千位符前两位，并在数字前显示人民币符号（Salary）。
- COLUMN COUNT(sal)…：设置的 COUNT(sal)题标为 Employee 并显示 3 位数字，中间对齐（Number）。
- COLUMN SUM(sal)…：设置的 SUM(sal)题标为 Summary 并显示 7 位数字，小数点后两位，千位符前两位，并在数字前显示人民币符号（Salary）。
- SET FEEDBACK ON：打开反馈。

我们将例 9-32 的命令存在 report.sql 的脚本文件中。该文件存在 d:\Oracle\ming 的目录下。请注意，必须在存文件之前用操作系统命令创建此目录。可以使用任何您喜欢的目录或文件名。之后装入并运行该脚本文件，如例 9-33 所示。

例 9-33

```
SQL> @d:\ Oracle\ming\report
```

例 9-33 结果

```
星期一 9月  09                                                第    1
                       ======= Sun_Moon IT Company ======
                        ===== Employee Salary Report =====

 Department  Job                 Average Employee               Summary
   Number    Category             Salary  Number                Salary
------------ ------------ ---------------- --------- --------- ------
          10 CLERK               RMB1,300.00           1         RMB1,300.00
             MANAGER             RMB2,450.00           1         RMB2,450.00
             PRESIDENT           RMB5,000.00           1         RMB5,000.00

          20 ANALYST             RMB3,000.00           2         RMB6,000.00
             CLERK                 RMB950.00           2         RMB1,900.00
             MANAGER             RMB2,975.00           1         RMB2,975.00

          30 CLERK                 RMB950.00           1           RMB950.00
             MANAGER             RMB2,850.00           1         RMB2,850.00
             SALESMAN            RMB1,400.00           4         RMB5,600.00

                            Strictly Confidential !!!
```

以下是产生例 9-33 所显示的报告的推荐步骤：

（1）建立所需的 SQL 语句。

（2）测试这些 SQL 语句。

（3）把经过测试并没有错误的 SQL 语句存在一个脚本文件中。

（4）把脚本文件装入正文编辑器。

（5）在 SQL 语句之前加上所需的 SQL*Plus 格式化命令。

（6）在 SQL 语句之后加上结束符“/”。
（7）之后恢复 SQL*Plus 环境变量的默认设置。
（8）存储该脚本文件。
（9）运行该脚本文件。

注意：

尽管所介绍的 SQL*Plus 格式化命令看上去很繁琐，而且想得到一个好看的显示输出需要花不少的时间来调整，但在软件开发时却很少直接使用这些格式化命令，因为现在市场上有很多图形界面的开发工具，利用这些开发工具可以比较容易地得到所需要的显示输出。但这些开发工具最后还是把鼠标的点击或拖拉等都转换成了 SQL*Plus 的格式化命令。因此，使用 SQL*Plus 的格式化命令来产生报告是最原始的方法。

9.7 数据字典和数据字典视图

数据字典是由 Oracle 服务器创建和维护的一组系统表。SYS 用户拥有所有的数据字典表。考虑到系统效率，Oracle 服务器以最简捷（最快）的方式来操作数据字典的基表，所以，数据字典的基表中所存的数据就像天书一样，几乎没什么人能看得懂。因此，很少有人直接访问这些基表。取而代之的是，绝大多数用户包括数据库管理员（DBA）在内都是通过访问数据字典视图来得到数据库的相关信息。数据字典视图把数据字典基表的信息转换成了人们较为容易理解的形式，它包括了用户名、用户的权限、对象名、约束和审计等方面的信息。

数据字典视图分为 3 大类。它们用前缀来区别，其前缀分别为 USER、ALL 和 DBA。许多数据字典视图包含相似的信息。为了帮助读者理解，我们利用图 9.1 来解释这些数据字典视图。

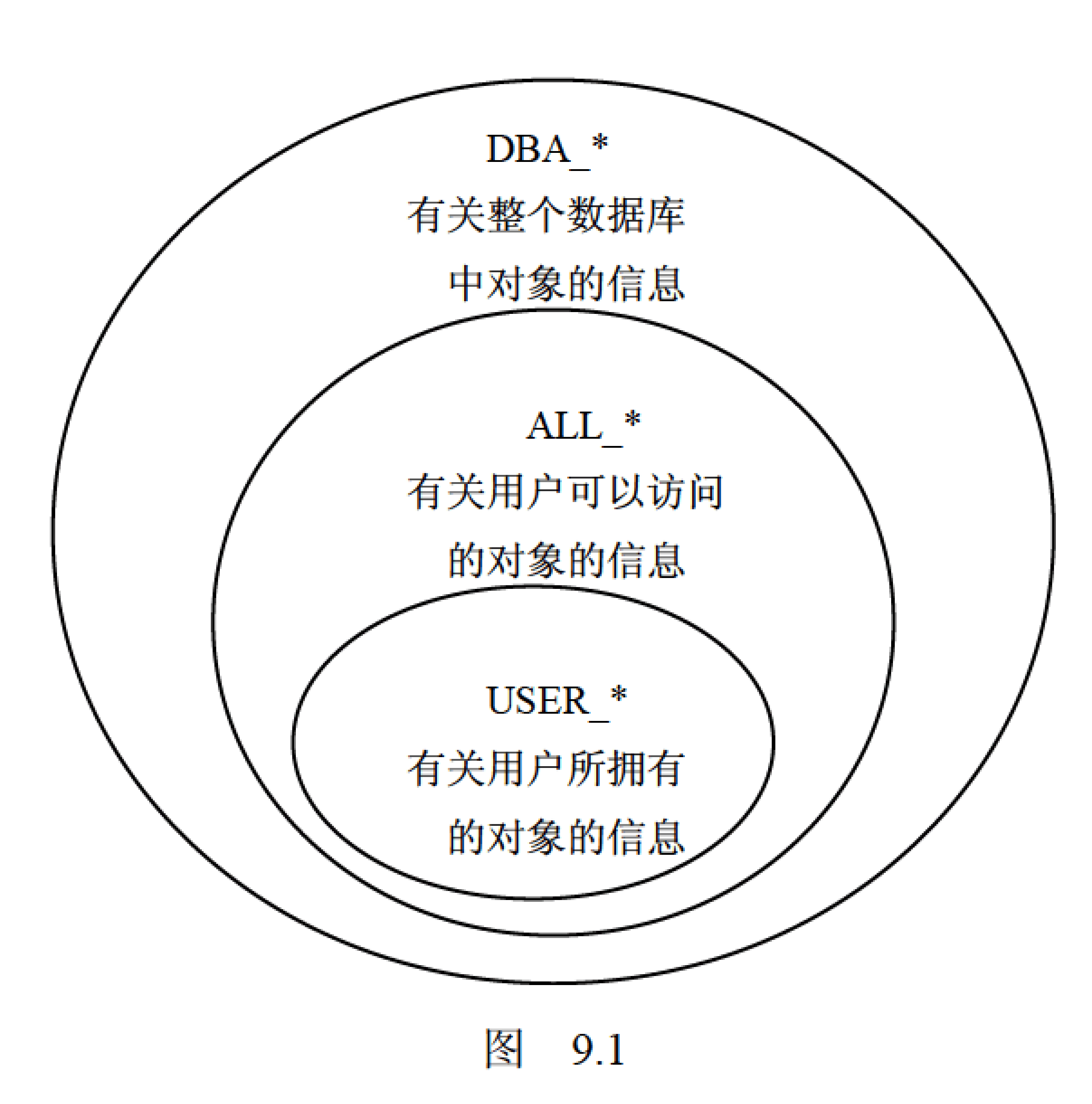


图 9.1

- USER_* ：有关用户所拥有的对象的信息，即用户自己创建的对象的信息。

- ALL_*　：有关用户可以访问的对象的信息，即用户自己创建的对象的信息再加上其他用户创建的对象但该用户有权访问的信息。
- DBA_*　：有关整个数据库中对象的信息。

这里的“*”可以为 TABLES、INDEXES 和 OBJECTS 等。

以前缀为 USER 开始的数据字典视图中的列与 ALL 和 DBA 中的列几乎是相同的，但是以前缀为 ALL 和 DBA 开始的数据字典视图比 USER 多了一列 OWNER。下面我们用数据字典*_OBJECTS 来演示它们之间的这一区别。首先我们必须以 DBA 用户连接数据库，因为只有 DBA 用户可以使用 DBA_*数据字典。您可以使用例 9-34 的命令来达到这一目的，或在启动 SQL*Plus 时以 SYSTEM 用户登录，该用户的口令为 MANAGER（如果没人修改过的话）。

例 9-34

```
SQL> connect system/manager
```

例 9-34 结果

```
已连接。
```

现在可以使用例 9-35~例 9-37 来分别显示 user_objects、all_objects 和 dba_objects 中的每一列的定义。

例 9-35

```
SQL> desc user_objects
```

例 9-35 结果

```
名称                                     是否为空? 类型
---------------------------------- --------- ----------------------------
OBJECT_NAME                                    VARCHAR2(128)
SUBOBJECT_NAME                                 VARCHAR2(30)
OBJECT_ID                                      NUMBER
DATA_OBJECT_ID                                 NUMBER
OBJECT_TYPE                                    VARCHAR2(18)
CREATED                                        DATE
LAST_DDL_TIME                                  DATE
TIMESTAMP                                      VARCHAR2(19)
STATUS                                         VARCHAR2(7)
TEMPORARY                                      VARCHAR2(1)
GENERATED                                      VARCHAR2(1)
SECONDARY                                      VARCHAR2(1)
```

例 9-36

```
SQL> desc all_objects
```

例 9-36 结果

```
名称                                     是否为空? 类型
----------------------- --------- ----------------------------
OWNER                                  NOT NULL VARCHAR2(30)
```

```
OBJECT_NAME                                      NOT NULL VARCHAR2(30)
SUBOBJECT_NAME                                            VARCHAR2(30)
OBJECT_ID                                        NOT NULL NUMBER
DATA_OBJECT_ID                                            NUMBER
OBJECT_TYPE                                               VARCHAR2(18)
CREATED                                          NOT NULL DATE
LAST_DDL_TIME                                    NOT NULL DATE
TIMESTAMP                                                 VARCHAR2(19)
STATUS                                                    VARCHAR2(7)
TEMPORARY                                                 VARCHAR2(1)
GENERATED                                                 VARCHAR2(1)
SECONDARY                                                 VARCHAR2(1)
```

例 9-37

```
SQL> desc dba_objects
```

例 9-37 结果

```
名称                                        是否为空? 类型
-------------------------------- ----------------------------
OWNER                                                     VARCHAR2(30)
OBJECT_NAME                                               VARCHAR2(128)
SUBOBJECT_NAME                                            VARCHAR2(30)
OBJECT_ID                                                 NUMBER
DATA_OBJECT_ID                                            NUMBER
OBJECT_TYPE                                               VARCHAR2(18)
CREATED                                                   DATE
LAST_DDL_TIME                                             DATE
TIMESTAMP                                                 VARCHAR2(19)
STATUS                                                    VARCHAR2(7)
TEMPORARY                                                 VARCHAR2(1)
GENERATED                                                 VARCHAR2(1)
SECONDARY                                                 VARCHAR2(1)
```

从例 9-35、例 9-36 和例 9-37 显示的结果可以看出，user_objects 中没有 OWNER 这一列，而 all_objects 和 dba_objects 中都包含了 OWNER 这一列。这是因为使用 user_objects 的用户是在查看自己的对象，他当然知道自己是谁，所以 OWNER 这一列也就没必要了。其他的 user_*、all_*和 dba_* 数据字典也遵守同样的规律，如*_tables 和*_indexes。

9.8 格式化数据字典视图的输出

假设您仍在 SYSTEM 用户下，现在想知道用户 SCOTT 所拥有的所有对象及它们的一些细节，可以使用例 9-38 的查询语句。

例 9-38

```
SQL> SELECT owner, object_name, object_id, created, status
  2  FROM   all_objects
  3  WHERE  owner = 'SCOTT';
```

例 9-38 结果

```
OWNER                          OBJECT_NAME                    OBJECT_ID
------------------------------ ------------------------------ -----
CREATED    STATUS
---------- -------
SCOTT                          BONUS                              32122
04-9月 -01 VALID

SCOTT                          DEPT                               32118
04-9月 -01 VALID

SCOTT                          EMP                                32120
04-9月 -01 VALID


OWNER                          OBJECT_NAME                    OBJECT_ID
------------------------------ ------------------------------ -----
CREATED    STATUS
---------- -------
SCOTT                          EMP_NULL                           32130
14-7月 -02 VALID

SCOTT                          MANAGER                            32137
13-8月 -02 VALID

SCOTT                          PK_DEPT                            32119
04-9月 -01 VALID

OWNER                          OBJECT_NAME                    OBJECT_ID
------------------------------ ------------------------------ ----
CREATED    STATUS
---------- -------
SCOTT                          PK_EMP                             32121
04-9月 -01 VALID

SCOTT                          SALGRADE                           32123
04-9月 -01 VALID
已选择 8 行。
```

从例 9-38 的显示结果可以看出，这一语句确实得到了我们所需要的结果，但其显示却很乱，很难看懂。还记得前面讲的吗？如果想得到好看的显示输出，要使用 SQL*Plus 的格式化命令。下面的例 9-39 使用了 3 个 COLUMN 命令来分别格式化列 owner、object_name 和 object_type。

例 9-39

```
SQL> col owner for a8
SQL> col object_name for a12
SQL> col object_type for a10
```

现在您可以再使用与例 9-39 的查询语句完全相同的例 9-40 来显示有关用户 SCOTT 所拥有的所有对象的信息。

例 9-40

```
SQL> SELECT owner, object_name, object_id, created, status, object_type
  2  FROM   all_objects
  3  WHERE  owner = 'SCOTT';
```

例 9-40 结果

```
OWNER    OBJECT_NAME   OBJECT_ID CREATED     STATUS  OBJECT_TYP
-------- ------------ ---------- ---------- -------- ----------
SCOTT    BONUS             32122 04-9 月 -01 VALID   TABLE
SCOTT    DEPT              32118 04-9 月 -01 VALID   TABLE
SCOTT    EMP               32120 04-9 月 -01 VALID   TABLE
SCOTT    EMP_NULL          32130 14-7 月 -02 VALID   TABLE
SCOTT    MANAGER           32137 13-8 月 -02 VALID   TABLE
SCOTT    PK_DEPT           32119 04-9 月 -01 VALID   INDEX
SCOTT    PK_EMP            32121 04-9 月 -01 VALID   INDEX
SCOTT    SALGRADE          32123 04-9 月 -01 VALID   TABLE
已选择 8 行。
```

很显然例 9-40 的显示结果非常清晰，也更容易理解。

9.9　如何使用数据字典视图

对数据字典视图介绍了这么多，它们到底对初学者有什么用处呢？如果您刚被某个公司聘为 Oracle 的工作人员，一定想知道在您的账号（用户名）下有哪些表。数据字典 user_tables 中就存有这些信息。可以用例 9-41 的查询语句来得到您所需要的信息（假设您现在是以 SCOTT 用户登录的）。

例 9-41

```
SQL> SELECT table_name
  2  FROM user_tables;
```

例 9-41 结果

```
TABLE_NAME
------------
BONUS
DEPT
EMP
EMP_NULL
MANAGER
SALGRADE
已选择 6 行。
```

注意：

在您的系统上显示的表可能会有所不同。

您也一定想知道有哪些表您可以使用，这时数据字典 all_tables 就派上用场了。例 9-42 的查询语句可以得到您所需要的信息。

例 9-42

```
SQL> SELECT table_name, owner
  2  FROM all_tables
  3  WHERE owner NOT LIKE '%SYS';
```

例 9-42 结果

```
TABLE_NAME                      OWNER
----------------------------  ------
DEF$_TEMP$LOB                  SYSTEM
HELP                           SYSTEM
DEPT                           SCOTT
EMP                            SCOTT
BONUS                          SCOTT
SALGRADE                       SCOTT
MANAGER                        SCOTT
EMP_NULL                       SCOTT
已选择 8 行。
```

另外一个可能会用到的是数据字典 user_catalog，您可以用例 9-43 的查询语句来得到该数据字典的结构。

例 9-43

```
SQL> desc user_catalog
```

例 9-43 结果

```
 名称                                   是否为空? 类型
 ----------------------------------------- -------- ------------
```

```
TABLE_NAME                                    NOT NULL VARCHAR2(30)
TABLE_TYPE                                             VARCHAR2(11)
```

从例 9-43 显示的结果可以看出，一个用户可用 user_catalog 看到他所拥有的所有表的名字和类型。与使用 user_tables 相比，也许使用 user_catalog 更简单，如例 9-44 所示。

例 9-44

```
SQL> select * from user_catalog;
```

例 9-44 结果

```
TABLE_NAME                     TABLE_TYPE
------------------------------ ----------
BONUS                          TABLE
DEPT                           TABLE
EMP                            TABLE
EMP_NULL                       TABLE
MANAGER                        TABLE
PLAN_TABLE                     TABLE
SALGRADE                       TABLE
已选择 7 行。
```

数据字典 user_catalog 有一个别名叫 cat，用户可以用它得到和 user_catalog 完全相同的信息，如例 9-45 所示。

例 9-45

```
SQL> select * from cat;
```

例 9-45 结果

```
TABLE_NAME                     TABLE_TYPE
------------------------------ ----------
BONUS                          TABLE
DEPT                           TABLE
EMP                            TABLE
EMP_NULL                       TABLE
MANAGER                        TABLE
PLAN_TABLE                     TABLE
SALGRADE                       TABLE
已选择 7 行。
```

☠ **注意：**

如果您是一个初学者，并且以下的内容您觉得很难理解的话，可以忽略这部分的内容。

如果您刚被某公司聘为 Oracle 数据库管理员，想知道您所管理的 Oracle 数据库的名字和创建日期等信息，可使用数据字典 v$database 来得到有关的信息。以 v$开始的数据字典为动态表，即 Oracle 服务器要随时修改它们，只有 Oracle 数据库管理员（DBA）可以访问这些数据字典。为了使用数据字典 v$database，您需要以 SYS 或 SYSTEM 用户身份登录

Oracle 数据库，如例 9-46 所示。

例 9-46

```
SQL> connect system/manager
```

您可以使用例 9-47 的查询语句来得到正在运行的数据库的名字、创建日期和运行的模式。

例 9-47

```
SQL> SELECT name, created, log_mode
   2  FROM v$database;
```

例 9-47 结果

```
NAME       CREATED     LOG_MODE
--------- ---------- ------------—
Oracle9i  11-7 月 -02 NOARCHIVELOG
```

例 9-47 显示的结果告诉我们，当前运行的数据库名字为 Oracle 9i，其创建日期为 2002 年 7 月 11 日，该数据库运行在非归档模式。

作为一名新的 Oracle 数据库管理员，您也应该知道有关 Oracle 实例的信息，可以使用例 9-48 的查询语句从数据字典 v$instance 中得到相关的信息。

例 9-48

```
SQL> SELECT instance_name, host_name, version, archiver
   2  FROM v$instance;
```

例 9-48 结果

```
INSTANCE_NAME   HOST_NAME            VERSION           ARCHIVE
--------------- -------------------- ----------------- -------
Oracle9i       CS-ZNAE5WTCSLHO      9.0.1.3.1          STOPPED
```

例 9-48 显示的结果告诉我们，当前运行的实例名也为 Oracle 9i，该实例运行在主机名为 CS-ZNAE5WTCSLHO 的电脑上，其版本为 9.0.1.3.1，归档后台进程没有启动。

另外也应该知道在您的系统上到底有多少用户和都是什么时候创建的，可以使用例 9-49 的查询语句来完成这一工作。

例 9-49

```
SQL> SELECT username, created
   2  FROM dba_users;
```

例 9-49 结果

```
USERNAME                       CREATED
------------------------------ ----------—
SYS                            04-9 月 -01
SYSTEM                         04-9 月 -01
DBSNMP                         04-9 月 -01
AURORA$JIS$UTILITY$            04-9 月 -01
```

```
AURORA$ORB$UNAUTHENTICATED 04-9 月 -01
SCOTT                      04-9 月 -01
…
```

例 9-49 显示的结果给出了系统中的所有用户和他们的创建日期。

9.10 小　　结

SET 和 SHOW 是一对控制 SQL*Plus 环境的命令。为了达到控制 SQL*Plus 环境的目的，使用 SET 命令来设置 SQL*Plus 的环境变量，使用 SHOW 命令来显示 SQL*Plus 的环境变量的配置。

为了产生优良的报告（输出），SQL*Plus 提供了一些格式化命令，其中使用最多的是 COLUMN 命令。即使在您所使用的 Oracle 数据库上没有安装任何图形工具的情况下，仍然可以用这些命令产生您想得到的显示输出。

Oracle 数据库上有许多数据字典，这些数据字典是由 Oracle 服务器创建和维护的。SYS 用户拥有所有的数据字典，它们存有许多的系统信息，如用户名、用户的权限、对象名、约束和审计等方面的信息。Oracle 用户和管理员可以通过查询某个或某几个数据字典来得到他们所需要的信息。

9.11 应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 什么是 SQL*Plus 的环境变量？
- 如何控制 SQL*Plus 的环境？
- SQL*Plus 的 SET 命令。
- SQL*Plus 的 SHOW 命令。
- 环境变量 ECHO。
- 环境变量 FEEDBACK。
- 其他常用的环境变量。
- COLUMN 命令。
- COLUMN 命令的格式化模式。
- 其他的常用格式化命令。
- 产生优良的报表的推荐步骤。
- 什么是数据字典视图？
- 数据字典视图的分类。
- 格式化数据字典视图的输出。
- 如何使用数据字典视图？
- 常用的数据字典视图。

第10章

创建表

在前面的第 9 章中我们经常使用表 dept 和表 emp 等，那么这些表是如何建立的呢？另外我们在第 7 章中所得到的 3 个三范式的表 ORDER、SUPPLIER 和 PRODUCT 又将如何存在于 Oracle 数据库中呢？下面我们就介绍如何创建表。

10.1　创建表的语句和例子

我们用 CREATE TABLE 语句来创建表。该语句为数据定义语言（Data Definition Language，DDL），它的格式如下：

CREATE TABLE [用户名.]表名

(列名 数据类型 [DEFAULT 默认值] [,…]);

您可以使用例 10-1 的 DDL 语句来创建表 product。该表是第 7 章中所得到的 3 个三范式表中的一个。

例 10-1

```
SQL> CREATE TABLE product
  2        (p_code      NUMBER(6),
  3         p_name      VARCHAR2(30),
  4         p_desc      VARCHAR2(100),
  5         P_price     NUMBER(5,2));
```

例 10-1 结果

```
表已创建。
```

下面您可以使用 SQL*Plus 的命令 DESC 来验证一下刚创建的表 product 是否正确，如例 10-2。

例 10-2

```
SQL> DESC product
```

例 10-2 结果

```
 名称                                    是否为空? 类型
 ----------------------------------------- -------- -------------
 P_CODE                                             NUMBER(6)
 P_NAME                                             VARCHAR2(30)
```

```
P_DESC                                     VARCHAR2(100)
P_PRICE                                    NUMBER(5,2)
```

从例 10-2 显示的结果可以看出，您所创建的表 product 准确无误。

如果您想在 Oracle 数据库中创建表的话，您必须具有 CREATE TABLE 的系统权限。这个权限是由 Oracle 数据库管理员（DBA）授予的。Oracle 数据库管理员使用 GRANT 语句把系统权限授予用户（在后面的章节中我们要进一步介绍系统权限的管理）。另外，您还必须有足够的磁盘空间。用户磁盘空间的大小是由 Oracle 数据库管理员（DBA）使用 CREATE USER/ALTER USER 语句中的 QUOTA 子句指定的（注：如何指定用户磁盘空间的大小是属于 Oracle ARCHITECTURE（体系结构）的内容。如果您对这一部分感兴趣，可以参阅 Oracle ARCHITECTURE（体系结构）中的有关用户管理一章）。

10.2 命名和引用规则

除了上面所述，如果您想在 Oracle 数据库中创建表的话，还必须指定表名、列名、列的数据类型及数据的长度。您有时也需要指定数据的默认值。

Oracle 数据库中的表和列名是以英文字母开头的字母数字序列，其命名遵循如下的规则：

- 必须以英文字母开头，之后跟大写或小写英文字母、数字字符、#、$、_。
- 名字的长度最短为一个字母，最长为 30 个字符。
- 一定不能与 Oracle 数据库系统的保留关键字相同，如例 10-3（注：以下 DDL 语句是在 SCOTT 用户下输入的）。

例 10-3

```
SQL> CREATE TABLE from (aa char(18));
```

例 10-3 结果

```
CREATE TABLE from (aa char(18))
             *
ERROR 位于第 1 行:
ORA-00903: 表名无效
```

根据例 10-3 显示结果的错误信息，您应该已经看出来了，错误的原因是例 10-3 的 DDL 语句中所用的表名 from 为 Oracle 数据库系统的保留关键字。

表名一定不能与同一用户下的任何其他的对象相同，如例 10-4（注：以下 DDL 语句也是在 SCOTT 用户下输入的）。

例 10-4

```
SQL> create table emp(xx number);
```

例 10-4 结果

```
create table emp(xx number)
             *
```

```
ERROR 位于第 1 行:
ORA-00955: 名称已由现有对象使用
```

根据例 10-4 显示结果的错误信息，您应该已经看出来了，错误的原因是例 10-4 的 DDL 语句中所用的表名 EMP 已在 SCOTT 用户中存在。

但是同一用户不同表的列名可以相同，如例 10-5 和例 10-6（注：以下 SQL*Plus 命令也是在 SCOTT 用户下输入的）。

例 10-5

```
SQL> DESC dept
```

例 10-5 结果

```
 名称                是否为空?   类型
 ---------- ---------   -------------
 DEPTNO                   NOT NULL  NUMBER(2)
 DNAME                              VARCHAR2(14)
 LOC                                VARCHAR2(13)
```

例 10-6

```
SQL> DESC emp
```

例 10-6 结果

```
 名称                是否为空? 类型
   ---------------- --------- -------------
 EMPNO                    NOT NULL NUMBER(4)
 ENAME                             VARCHAR2(10)
 JOB                               VARCHAR2(9)
 MGR                               NUMBER(4)
 HIREDATE                          DATE
 SAL                       NUMBER(7,2)
 COMM                      NUMBER(7,2)
 DEPTNO                    NUMBER(2)
```

从例 10-5 和例 10-6 显示的结果可以看出，在 dept 表和 emp 表中都包含了 DEPTNO 这一列。

以上我们介绍了表和列的命名规则，那么如何引用表呢？其实在前面的章节中我们已经多次引用了不同的表，但都只是引用用户自己的表。在那些例子中，我们只是在关键字 FROM 之后给出所需的表名，如例 10-7 的查询语句。

例 10-7

```
SQL> SELECT *
  2  FROM dept;
```

例 10-7 结果

```
    DEPTNO DNAME          LOC
---------- -------------- --------
```

```
        10 ACCOUNTING      NEW YORK
        20 RESEARCH        DALLAS
        30 SALES           CHICAGO
        40 OPERATIONS      BOSTON
```

现在我们来介绍如何引用属于其他用户的表。为了说明如何引用属于其他用户的表，您可以用例 10-8 的命令由当前的 SCOTT 用户转换为 SYSTEM 用户，其中 manager 为 SYSTEM 用户的口令。

例 10-8

```
SQL> connect system/manager
```

例 10-8 结果

```
已连接。
```

如果您想得到 dept 表中的信息，您也许会试着使用例 10-9 的查询语句。

例 10-9

```
SQL> SELECT *
2  FROM dept;
```

例 10-9 结果

```
FROM dept
     *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

例 10-9 显示的结果告诉我们，dept 表不属于 SYSTEM 用户。我们知道该表属于 SCOTT 用户，您应该在表名前加上用户名，用户名和表名之间用“.”隔开。现在您可以重写例 10-9 的查询语句，如例 10-10。

例 10-10

```
SQL> SELECT *
    2  FROM scott.dept;
```

例 10-10 结果

```
    DEPTNO DNAME          LOC
---------- -------------- --------
        10 ACCOUNTING      NEW YORK
        20 RESEARCH        DALLAS
        30 SALES           CHICAGO
        40 OPERATIONS      BOSTON
```

从例 10-10 可以看出引用属于其他用户的表的格式为：

用户名.表名

有些书中使用了模式（schema）名而不是用户名。其实，用户名与模式名在 Oracle 系

统中是没有区别的，只是模式的学术味更浓一些。在本书中我们对这两个术语是不加区分的，而且更倾向于使用用户名，因为我个人认为对一般用户来讲用户要比模式更容易理解。

10.3　列的数据类型和默认值

在明白了列的命名规则之后，您一定想知道列的数据类型是如何定义的以及列可以定义为哪些数据类型？在 Oracle 数据库中经常使用的数据类型有以下 4 种。

- VARCHAR2(size)：变长字符型数据。其中，size 为该列最多可容纳的字符个数，必须定义，它的默认值和最小值均为 1，最大值为 4000。
- CHAR(size)：定长字符型数据。其中，size 为该列最多可容纳的字符个数，它的默认值和最小值均为 1，最大值为 2000。
- DATE：日期型数据。其中，日期和时间的取值范围是从公元前 1471 年 1 月 1 日到公元 9999 年 12 月 31 日。
- NUMBER(p,s)：数字型数据。其中，p 为十进制数的总长度（位数），s 为十进制数小数点后面的位数，p 的最小值为 1，最大值为 38，s 的最小值为−84，最大值为 124。

如果您定义了表中的某一列为变长字符型数据（VARCHAR2），那么当您插入或修改该列时，Oracle 只把实际的字符存入数据库中，如在例 10-1 中的 product 表中有如下的定义：

p_desc VARCHAR2(100)

虽然您定义 p_desc 的最大长度为 100 个字符，但是如果您在向该列输入数据时输入的为 NEW 这 3 个字符，Oracle 实际上只把 NEW 这 3 个字符存入到数据库中。所以使用变长字符型数据（VARCHAR2）会节省空间，特别是当所定义的列的取值范围变化很大时。

如果您定义了表中的某一列为定长字符型数据（CHAR），那么当您插入或修改该列时，Oracle 存入数据库中的字符数是由 size 所定义的，没有数据的部分由 Oracle 填上空格。如果您将例 10-1 中 p_desc 的定义改写为：

p_desc CHAR(100)

那么，此时您定义 p_desc 的最大长度为 100 个字符。如果向这一列输入数据时，如输入的为 NEW，只有 3 个字符，Oracle 把 NEW 这 3 个字符再加上 97 个空格，共 100 个字符存入到数据库中。所以使用定长字符型数据（CHAR）会浪费一些空间，特别是当所定义的列的取值范围变化很大时，浪费的空间会很大。

一般来讲，变长字符型数据（VARCHAR2）节省空间，但 Oracle 处理的速度可能较慢。而定长字符型数据（CHAR）会浪费一些空间，但 Oracle 处理它的速度可能快一些。对于大多数数据库应用来说，变长字符型数据（VARCHAR2）要比定长字符型数据（CHAR）更有效。但对于那些长度没有变化的列，定长数据类型（CHAR）是最好的选择。例如，性别（gender）只有两个选择：男（M-MALE）或女（F-FEMALE），每种选择都可以用一个字符表示。

除了前面介绍的 4 种常用的数据类型外，为了处理多媒体对象（Large Object，LOB），

Oracle 提供了如下的 LOB 数据类型。

- CLOB 数据类型（Character Large Object）：用于在数据库中存储单字节的大数据对象，如讲演稿或简历等。
- BLOB 数据类型（Binary Large Object）：用于在数据库中存储大的二进制对象，如照片或幻灯片等。
- CLOB 和 BLOB 数据类型的列中，许多操作是不能直接使用 Oracle 的数据库命令来完成的，因此 Oracle 提供了一个叫 DBMS_LOB 的 PL/SQL 软件包来维护 LOB 数据类型的列。
- BFILE 数据类型（Binary File）：用于在数据库外的操作系统文件中存储大的二进制对象，如电影胶片等。BFILE 数据类型是外部数据类型，因此定义为 BFILE 数据类型的列是不能通过 Oracle 的数据库命令来操作的，这些列只能通过操作系统命令或第三方软件来维护。

为了提高效率，Oracle 还提供 RAW 数据类型。

- RAW 数据类型：在数据库中直接存储二进制数据。此种类型的数据占用的存储空间小，操作效率也高。但在网络环境中不同的计算机上传输资料时，Oracle 服务器是不进行任何的字符集转换的。

为了和以前的 Oracle 版本兼容，Oracle 继续支持 LONG 和 LONG RAW 数据类型。

- LONG 和 LONG RAW 数据类型：主要用于在 Oracle 8 以前的数据库中存储无结构的数据，如二进制图像、文本件和地理信息等。

在 Oracle 8 以后的版本，LOB 数据类型可以完全取代 LONG 数据类型，而且 Oracle 服务器操作 LOB 数据类型比操作 LONG 数据类型效率更高。另外，您只能在一个表中定义一个 LONG 数据类型的列，但可以在一个表中定义任意多个 LOB 数据类型的列。LONG 数据类型的列最多可以存储 2GB 的数据，而 LOB 数据类型的列最多可以存储 4GB 的数据。

☠ **注意：**

这里的 LOB 包括了 CLOB、BLOB 等数据类型，LONG 包括了 LONG 和 LONG RAW 数据类型。

10.4 创建表的例子

现在您可以使用在这一章中刚讲过的 DDL 语句来创建第 7 章中的另外两个表 supplier 和 ord（er），其 DDL 语句如例 10-11 和例 10-12。

例 10-11

```
SQL> CREATE TABLE supplier
  2        ( s_code   NUMBER(6),
  3          sname    VARCHAR2(25),
  4          contact  VARCHAR2(15),
  5          phone    VARCHAR2(15),
  6          fax      VARCHAR2(15));
```

例 10-11 结果

```
表已创建。
```

例 10-12

```
SQL> CREATE TABLE ord
  2         (ordno    NUMBER(8),
  3          p_code   NUMBER(6),
  4          s_code   NUMBER(6),
  5          ordate   DATE,
  6          unit     NUMBER(6),
  7          price    NUMBER(8,2));
```

例 10-12 结果

```
表已创建。
```

创建了这两个表之后，您可使用 SQL*Plus 的命令 DESC 来检验它们是否正确，如例 10-13 和例 10-14。

例 10-13

```
SQL> DESC supplier
```

例 10-13 结果

```
名称                                   是否为空? 类型
-----------------------------  --------  ------------
S_CODE                                          NUMBER(6)
SNAME                                           VARCHAR2(25)
CONTACT                                         VARCHAR2(15)
PHONE                                           VARCHAR2(15)
FAX                                             VARCHAR2(15)
```

例 10-14

```
SQL> DESC ord
```

例 10-14 结果

```
名称                                   是否为空? 类型
-----------------------------------------  --------  ---
ORDNO                                           NUMBER(8)
P_CODE                                          NUMBER(6)
S_CODE                                          NUMBER(6)
ORDATE                                          DATE
UNIT                                            NUMBER(6)
PRICE                                           NUMBER(8,2)
```

现在您也可以使用数据字典 cat 来检验它们是否存在，其 SQL 语句如例 10-15 所示。

例 10-15

```
SQL> SELECT *
  2  FROM cat;
```

例 10-15 结果

```
TABLE_NAME                     TABLE_TYPE
------------------------------ -----------
BONUS                          TABLE
DEPT                           TABLE
EMP                            TABLE
ORD                            TABLE
PRODUCT                        TABLE
SALGRADE                       TABLE
SUPPLIER                       TABLE
```

注意：

需要说明的是，在这一章中所创建的这 3 个表都不是关系数据库的表，因为它们都没有定义主键（Primary Key）。

10.5 利用子查询来创建表

假设您所在的公司在其他国家或城市有许多个分公司，这些分公司的一些分析员要使用统计软件对 emp 表中的一些列进行大规模的统计分析，这些分析可能需要反复查询 emp 表，假设 emp 表存有一百多万行数据，如果这些分析员都访问总公司的 Oracle 服务器上的表 emp 的话，网络的流量可能会非常大，这样对网络和总公司的 Oracle 服务器的压力都会很大，系统的整体效率会大幅度下降。

在这种情况下，您可以为他们生成一个临时的表 worker，该表只包含了他们所需要的列和记录。之后把这个表传到他们本地的计算机上，这样可以在很大程度上减轻网络和总公司的 Oracle 服务器的压力，从而使系统的整体效率大为改进。您可以使用类似于例 10-16 的 DDL 命令来建立 worker 表。

例 10-16

```
SQL> CREATE TABLE worker
  2  AS
  3  SELECT empno, ename name, job, sal+NVL(comm,0) income
  4  FROM emp
  5  WHERE job NOT IN ('MANAGER', 'PRESIDENT');
```

例 10-16 结果

```
表已创建。
```

现在您可以使用 SQL*Plus 的命令 DESC 来查看一下表 worker 的结构是否正确，如例 10-17 所示。

例 10-17

```
SQL> DESC worker
```

例 10-17 结果

```
名称                                              是否为空? 类型
------------------------------------------------- -------- ----
EMPNO                                                      NUMBER(4)
NAME                                                       VARCHAR2(10)
JOB                                                        VARCHAR2(9)
INCOME                                                     NUMBER
```

最后您可以使用如例 10-18 的查询语句来验证表 worker 中的资料是否正确。

例 10-18

```
SQL> SELECT *
  2  FROM worker;
```

例 10-18 结果

```
    EMPNO NAME       JOB           INCOME
---------- ---------- --------- ----------
     7369 SMITH      CLERK            800
     7499 ALLEN      SALESMAN        1900
     7521 WARD       SALESMAN        1750
     7654 MARTIN     SALESMAN        2650
     7788 SCOTT      ANALYST         3000
     7844 TURNER     SALESMAN        1500
     7876 ADAMS      CLERK           1100
     7900 JAMES      CLERK            950
     7902 FORD       ANALYST         3000
     7934 MILLER     CLERK           1300
已选择 10 行。
```

使用子查询来创建表的语句格式如下：

CREATE TABLE 表名
　　　　　　[(列名，列名…)]
AS 子查询;

使用子查询来创建表时需要注意如下事项：

- Oracle 使用您所指定的列和表名来创建表，所有的行（记录）是由子查询得到的，并插入到该表中。
- 所说明的列名的个数一定要和子查询列的个数相同。
- 所说明的每一列的数据类型必须与子查询的每一列的数据类型相匹配。

- 所说明的列名必须符合列的命名规则。
- 如果没有给出列名，列名与子查询中的列名相同，而且列的个数和子查询列的个数也相同。

10.6 修改表的结构

由于商业环境是不断变化的，客户的需求也在不断变化，所以当一个表建立后，经过一段时间的使用，它的结构就可能需要变化。有一位国外的著者写到“在现代的社会里唯一不变的是变这个字”。所以在现代的社会里，永远也不要梦想设计一个千秋万代永远不变的完美的系统。但 Oracle 给这种变化提供了方便。您可以使用 ALTER TABLE 语句来修改表的结构。Oracle 在 ALTER TABLE 语句中提供了以下 4 种修改表结构的子语。

（1）在一个表中加入一个新的列。它的格式如下：

```
ALTER TABLE 表名
ADD        (列名  数据类型   [ DEFAULT   表达式 ]
           [,列名  数据类型  ]...);
```

您没有办法指定您所加入的列的位子，新的列永远为最后一列。

您可以使用如下的 ALTER TABLE 语句（例 10-19）在 worker 表中加入 hiredate 这一列，该列的数据类型为日期型。

例 10-19

```
SQL> ALTER TABLE worker
   2  ADD        (hiredate DATE);
```

例 10-19 结果

```
表已更改。
```

现在您可以使用例 10-20 的查询语句来检查您刚做的 DDL 操作是否正确。

例 10-20

```
SQL> SELECT *
   2  FROM worker;
```

例 10-20 结果

```
     EMPNO NAME       JOB            INCOME HIREDATE
---------- ---------- --------- ---------- -------
      7369 SMITH      CLERK            800
      7499 ALLEN      SALESMAN        1900
      7521 WARD       SALESMAN        1750
      7654 MARTIN     SALESMAN        2650
      7788 SCOTT      ANALYST         3000
      7844 TURNER     SALESMAN        1500
```

```
      7876  ADAMS       CLERK          1100
      7900  JAMES       CLERK           950
      7902  FORD        ANALYST        3000
      7934  MILLER      CLERK          1300
已选择 10 行。
```

（2）修改在一个表中已经存在的列，其格式如下：

ALTER TABLE 表名
MODIFY　(列名　数据类型　[DEFAULT　表达式]
　　　　[,列名　数据类型]...);

修改一个表中已经存在的列要注意以下的事项：

- 可以增加字符类型的列的宽度。
- 可以增加数字类型的列的宽度和精度。
- 只有当所有列的值都为空或者表中没有数据时，才可以减少列的宽度。
- 只有当所有列的值都为空时，才可以改变某一列的数据类型。
- 如果改变某一列的默认值，该默认值只影响以后的操作。
- 只有当某一列的值为空或者没有改变该列大小的情况下，才可以把 CHAR 类型的列改变成 VARCHAR2 类型的列，或把 VARCHAR2 类型的列改变成 CHAR 类型的列。

例 10-21 是一个修改 worker 表中 hiredate 列的默认值的例子。

例 10-21

```
SQL> ALTER TABLE worker
  2  MODIFY (hiredate DEFAULT SYSDATE);
```

例 10-21 结果

```
表已更改。
```

修改了 worker 表中 hiredate 列的默认值之后，您可以使用例 10-22 的查询语句来检查 HIREDATE 列的变化。

例 10-22

```
SQL> SELECT *
  2  FROM worker;
```

例 10-22 结果

```
     EMPNO  NAME        JOB          INCOME  HIREDATE
---------- ---------- ---------- ---------- --------
      7369  SMITH       CLERK           800
      7499  ALLEN       SALESMAN       1900
      7521  WARD        SALESMAN       1750
      7654  MARTIN      SALESMAN       2650
      7788  SCOTT       ANALYST        3000
```

```
      7844  TURNER     SALESMAN        1500
      7876  ADAMS      CLERK          1100
      7900  JAMES      CLERK           950
      7902  FORD       ANALYST        3000
      7934  MILLER     CLERK          1300
已选择 10 行。
```

从例 10-22 的查询结果可以看出，尽管您把 HIREDATE 列的默认值修改为系统的当前日期 SYSDATE，但是 worker 表中 HIREDATE 列的值没有任何变化，仍然为空。

现在您向 worker 表中插入一行数据，其中 HIREDATE 这一列为默认值，如例 10-23。

例 10-23

```
SQL> INSERT INTO worker(empno, name, job, income, hiredate)
  2  VALUES     (9000,  'MARY','CLERK', 1000, DEFAULT);
```

例 10-23 结果

```
已创建 1 行。
```

注意：

INSERT INTO 语句为 DML 语句，我们将在以后的章中详细的介绍。读者现在只要按照上面的语句在 SQL*Plus 中输入即可。

现在您再使用与例 10-22 相同的查询语句来检查 HIREDATE 列的变化，如例 10-24 所示。

例 10-24

```
SQL> SELECT *
  2  FROM worker;
```

例 10-24 结果

```
     EMPNO  NAME       JOB            INCOME  HIREDATE
----------  ---------- ---------  ----------  ----------
      7369  SMITH      CLERK           800
      7499  ALLEN      SALESMAN       1900
      7521  WARD       SALESMAN       1750
      7654  MARTIN     SALESMAN       2650
      7788  SCOTT      ANALYST        3000
      7844  TURNER     SALESMAN       1500
      7876  ADAMS      CLERK          1100
      7900  JAMES      CLERK           950
      7902  FORD       ANALYST        3000
      7934  MILLER     CLERK          1300
      9000  MARY       CLERK          1000  10-11 月-02
已选择 11 行。
```

从例 10-24 的查询结果可以看出，对于修改了 HIREDATE 列的默认值之后输入的记录，

其 HIREDATE 列为系统的当前日期 SYSDATE。

如果还是不放心的话，您可以再向 worker 表中插入一行数据，其中 NAME 这一列为"王老五"，而 HIREDATE 这一列仍为默认值，如例 10-25 所示。

例 10-25

```
SQL> INSERT INTO worker(empno, name, job, income, hiredate)
   2  VALUES     (9000,  '王老五' ,'CLERK', 600, DEFAULT);
```

例 10-25 结果

```
已创建 1 行。
```

现在再使用与例 10-22 相同的查询语句来检查 HIREDATE 列的变化，如例 10-26。

例 10-26

```
SQL> SELECT *
2  FROM worker;
```

例 10-26 结果

```
     EMPNO  NAME        JOB              INCOME  HIREDATE
---------- ---------- ---------- ---------- ----------
      7369  SMITH         CLERK              800
      7499  ALLEN         SALESMAN          1900
      7521  WARD          SALESMAN          1750
      7654  MARTIN        SALESMAN          2650
      7788  SCOTT         ANALYST           3000
      7844  TURNER        SALESMAN          1500
      7876  ADAMS         CLERK             1100
      7900  JAMES         CLERK              950
      7902  FORD          ANALYST           3000
      7934  MILLER        CLERK             1300
      9000  MARY          CLERK             1000  10-11 月-02
      9000  王老五        CLERK              600  10-11 月-02
已选择 12 行。
```

从例 10-26 的查询结果可以看出，刚输入的记录的 HIREDATE 列确实也是系统的当前日期 SYSDATE。

（3）从一个表中删除一列，其格式如下：

ALTER TABLE 表名

DROP COLUMN　列名;

删除一个表中已经存在的列要注意以下的事项：

- 使用以上的 ALTER TABLE 语句，一次只能删除一列。
- 在使用以上的 ALTER TABLE 语句删除了一列之后，该表中必须至少还有一列。
- 因为 ALTER TABLE 语句是 DDL 语句，所以所删除的列是无法被恢复的。
- 所删除的列可以包含数据，也可以不包含数据。

- 该语句只能在 Oracle 8i 及以上的版本上使用。

现在可以使用例 10-27 的 DDL 语句来删除 worker 表中的 HIREDATE 列。

例 10-27

```
SQL> ALTER TABLE worker
  2  DROP COLUMN hiredate;
```

例 10-27 结果

```
表已更改。
```

之后您可以再一次使用与例 10-22 相同的查询语句来检查 worker 表的变化，如例 10-28。

例 10-28

```
SQL> SELECT *
  2  FROM worker;
```

例 10-28 结果

```
    EMPNO NAME       JOB           INCOME
---------- ---------- --------- ----------
     7369 SMITH      CLERK            800
     7499 ALLEN      SALESMAN        1900
     7521 WARD       SALESMAN        1750
     7654 MARTIN     SALESMAN        2650
     7788 SCOTT      ANALYST         3000
     7844 TURNER     SALESMAN        1500
     7876 ADAMS      CLERK           1100
     7900 JAMES      CLERK            950
     7902 FORD       ANALYST         3000
     7934 MILLER     CLERK           1300
已选择 10 行。
```

从例 10-28 的查询结果可以看出，HIREDATE 列已经从 worker 表中删除了。

注意：

如果细心观察例 10-28 的查询结果，您就可以发现之前插入的两行记录并未在 worker 表中。这一问题涉及事务处理，我们将在下面的章节中详细讨论。

在一个表中删除一列，特别是在一个大表中删除一列是相当耗时的，对系统的效率冲击也很大，所以应尽可能地避免在数据库繁忙期间使用上述 DDL 语句。如果现在数据库特别繁忙，而就在此时您的老板让您立即删除某一个大表中的一列。您该如何处理呢？Oracle 提供了一个折中的方案，就是在 ALTER TABLE 语句中使用 SET UNUSED 子句。

（4）在一个表中把某一列置成无用（UNUSED），其格式如下：

ALTER TABLE 表名

SET UNUSED(列名);或

ALTER TABLE 表名

SET UNUSED COLUMN　列名;

当数据库空闲时，您再利用以下的 DDL 语句来删除已设置为无用（UNUSED）的列。

ALTER TABLE 表名

DROP UNUSED COLUMNS

使用 SET UNUSED 把表中的一列设置成无用（UNUSED）要注意以下的事项：

- 该选项只能在 Oracle 8i 及以上的版本上使用。
- 该选项只是将设置成无用的列标上记号，并不真正删除这一列。
- 由该选项设置成无用的列，无法用 SQL*Plus 命令或 SQL 语句看到。
- Oracle 把设置成无用的列当作删除列处理。
- 可以把一列、也可以把多列设置成无用。
- 可以使用 DROP(UNUSED)列名选项来删除被设置成无用的列。
- 因为该语句是一个 DDL 语句，所以没有恢复无用列的命令。

现在您可以使用例 10-29 的 DDL 语句把 worker 表中的 INCOME 列设置成无用。

例 10-29

```
SQL> ALTER TABLE worker
   2  SET   UNUSED (income);
```

例 10-29 结果

```
表已更改。
```

之后您可以使用 SQL*Plus 的命令 DESC 来查看表 worker 的结构，如例 10-30 所示。

例 10-30

```
SQL> DESC worker
```

例 10-30 结果

```
 名称                                   是否为空? 类型
 ------------------------------------- --------- ---------
 EMPNO                                            NUMBER(4)
 NAME                                             VARCHAR2(10)
 JOB                                              VARCHAR2(9)
```

从例 10-30 的显示结果可以看出，worker 表中的 INCOME 列已经不见了。

现在您可以再次使用与例 10-22 相同的查询语句来检查 worker 表的变化，如例 10-31 所示。

例 10-31

```
SQL> SELECT *
FROM worker;
```

例 10-31 结果

```
     EMPNO NAME       JOB
---------- ---------- ---------
```

```
      7369 SMITH      CLERK
      7499 ALLEN      SALESMAN
      7521 WARD       SALESMAN
      7654 MARTIN     SALESMAN
      7788 SCOTT      ANALYST
      7844 TURNER     SALESMAN
      7876 ADAMS      CLERK
      7900 JAMES      CLERK
      7902 FORD       ANALYST
      7934 MILLER     CLERK
已选择 10 行。
```

从例 10-31 结果可以看出，worker 表中的 INCOME 列中的数据也不见了。但是这时的 INCOME 列并没有被真正删除，只是被 Oracle 服务器标上了标志。等到系统平静时您可以使用例 10-32 的语句来把这一列从 worker 表中真正地删除掉。

例 10-32

```
SQL> ALTER TABLE worker
  2  DROP  UNUSED COLUMNS;
```

例 10-32 结果

```
表已更改。
```

10.7 改变对象的名字

什么是数据库中的对象？Oracle 数据库中的对象是一个存储在 Oracle 数据库中的数据结构。一个 Oracle 数据库中可以存有多种对象，这些对象的定义被存储在 Oracle 数据库的数据字典中。常用的对象有以下 5 种。

- 表（Table）：存储数据的基本单位，由行和列组成。
- 索引（Index）：为了改进某些查询性能的数据结构。
- 视图（View）：来自一个或多个表的数据子集。
- 序列（Sequence）：数值生成器。
- 同义词（别名）（Synonym）：赋予对象另外的名字。

我们已经详细地介绍了表（Table）对象。其他的 4 种对象将在后续的几章中详细介绍。除了以上提到的 5 种对象外，Oracle 数据库中还包括存储过程（Procedure）、函数（Function）和触发器（Trigger）等对象，这些内容将在 PL/SQL 程序设计的课程中介绍。

如果您在创建某一个对象时考虑不周，对象的名字取得不合适，您可以使用 Oracle 的 RENAME 语句来修改对象的名字。RENAME 语句的格式如下：

RENAME　对象原来的名字　对象现在的名字;

只有对象的主人才可以修改对象的名字。现在您可以使用例 10-33 的语句将表 worker 改为 staff。

例 10-33

```
SQL> RENAME worker to staff;
```

例 10-33 结果

```
表已重命名。
```

如果想用 SQL*Plus 的命令 DESC 来检查表 worker 的结构，您会得到对象 worker 不存在的出错提示，如例 10-34 所示。

例 10-34

```
SQL> DESC worker
```

例 10-34 结果

```
ERROR:
ORA-04043: 对象 worker 不存在
```

现在用同样的命令来检查表 staff 的结构，您就会发现它的结构与原来的 worker 表完全一样，如例 10-35 所示。

例 10-35

```
SQL> DESC staff
```

例 10-35 结果

```
 名称                                     是否为空? 类型
 ---------------------------------------- --------- --------
 EMPNO                                              NUMBER(4)
 NAME                                               VARCHAR2(10)
 JOB                                                VARCHAR2(9)
```

如果使用例 10-36 所示的查询语句，您会发现该查询所返回的结果与例 10-31 的结果完全相同。

例 10-36

```
SQL> SELECT *
  2  FROM staff;
```

例 10-36 结果

```
     EMPNO NAME       JOB
---------- ---------- --------
      7369 SMITH      CLERK
      7499 ALLEN      SALESMAN
      7521 WARD       SALESMAN
      7654 MARTIN     SALESMAN
      7788 SCOTT      ANALYST
      7844 TURNER     SALESMAN
      7876 ADAMS      CLERK
```

```
      7900  JAMES      CLERK
      7902  FORD       ANALYST
      7934  MILLER     CLERK
已选择 10 行。
```

注意：

如果您修改了一个对象的名字，那么使用该对象的软件或对象需要重新编译或修改，这可能会对系统的效率产生冲击。

10.8 为表和列加注释

按照软件工程的设计方法，当您在设计和开发一个软件系统时，您应该写下完整的文档和注释。Oracle 允许您使用 COMMENT 语句来为表或列添加注释。COMMENT 语句的格式如下：

COMMENT ON TABLE 表名 | COLUMN 表名.列名
IS '正文';

您可以使用例 10-37 的 SQL 语句在 staff 表上添加注释。

例 10-37

```
SQL> COMMENT ON TABLE staff
  2  IS 'Are you believed comments can help you to understand the designs?';
```

例 10-37 结果

```
注释已创建。
```

现在可以使用例 10-38 的查询来查看您刚在 staff 表上添加的注释。

例 10-38

```
SQL> SELECT comments
  2  FROM user_tab_comments
  3  WHERE table_name = 'STAFF';
```

例 10-38 结果

```
COMMENTS
------------------------------------------------------------------
Are you believed comments can help you to understand the designs?
```

您也可以使用例 10-39 的查询语句在 staff 表中的 job 列上添加注释。

例 10-39

```
SQL> COMMENT ON COLUMN staff.job
  2  IS 'If your answer is yes, you will meet a big trouble';
```

例 10-39 结果

```
注释已创建。
```

现在可以使用如例 10-40 的查询来查看您刚在 staff 表中的 job 列上添加的注释。

例 10-40

```
SQL> SELECT comments
  2  FROM user_col_comments
  3  WHERE table_name = 'STAFF'
  4  AND   column_name = 'JOB';
```

例 10-40 结果

```
COMMENTS
--------------------------------------------------
If your answer is yes, you will meet a big trouble
```

Oracle 没有提供如何从数据库中删除注释的语句，但您可以通过加入空串的方式从数据库中删除一条注释。您可以使用例 10-41 的语句来删除 staff 表中的 job 列上的注释。

例 10-41

```
SQL> COMMENT ON COLUMN staff.job
  2  IS '';
```

例 10-41 结果

```
注释已创建。
```

现在可以使用例 10-42 的查询语句来查看您刚删除的注释是否还存在？

例 10-42

```
SQL> SELECT comments
  2  FROM user_col_comments
  3  WHERE table_name = 'STAFF'
  4  AND   column_name = 'JOB';
```

例 10-42 结果

```
COMMENTS
----------------------------------------------------------------------
```

从例 10-42 的查询结果可以清楚地看出，staff 表中 job 列上的注释已被删除。

10.9 截断表和删除表

当一个表中的数据已经不再需要时，可以使用 TRUNCATE TABLE 语句将它们全部删除（截断），该语句为 DDL 语句。

TRUNCATE TABLE 语句的格式如下：

TRUNCATE TABLE 表名;

TRUNCATE TABLE 语句有如下的特性：

- 它删除表中所有的数据行，但保留表的结构。

- 如果没有备份的话，所删除的数据行无法恢复。
- 该语句释放表所占用的磁盘空间。
- 它并不触发（运行）表的删除触发器。

如果您不但要删除表中的数据，而且还要删除表的结构，您应该使用 DROP TABLE 语句，该语句也为 DDL 语句。

DROP TABLE 语句的格式如下：

DROP TABLE 表名;

DROP TABLE 语句有如下的特性：

- 它删除表中所有的数据行和表的结构。
- 它也删除表的所有索引（Indexes）。
- 如果没有备份的话，所删除的表无法恢复。
- 它提交所有的挂起事务（我们将在后面章节中介绍事务处理）。
- 所有基于该表的视图（Views）和别名（Synonyms）依然保留但已无效。

☠ **注意：**

TRUNCATE TABLE 和 DROP TABLE 语句是非常危险的语句，因为如果使用不当，可能会丢失大量宝贵的数据。尽管如此，Oracle 还假设使用这两个语句的人是专家，即使用这些语句的人知道他们在做什么和所产生的后果。Oracle 只检查用户的权限和语句的语法，如果它们没有问题，Oracle 就会忠实地执行用户所发的语句而且没有任何提示。因此建议在使用这两个语句之前最好做备份。

下面通过例子来演示 TRUNCATE TABLE 和 DROP TABLE 语句的功能和它们之间的差别。为了演示方便和安全起见，我们先按例 10-43 创建一个表 staff1。

例 10-43

```
SQL> CREATE TABLE staff1
  2  AS
  3  SELECT *
  4  FROM staff;
```

例 10-43 结果

```
表已创建。
```

现在可以使用如例 10-44 的查询语句来查看您刚创建的表 staff1 中的内容，您会发现它的内容与原来的 staff 表完全一样。

例 10-44

```
SQL> SELECT *
  2  FROM staff1;
```

例 10-44 结果

```
     EMPNO NAME       JOB
---------- ---------- --------
```

```
      7369 SMITH      CLERK
      7499 ALLEN      SALESMAN
      7521 WARD       SALESMAN
      7654 MARTIN     SALESMAN
      7788 SCOTT      ANALYST
      7844 TURNER     SALESMAN
      7876 ADAMS      CLERK
      7900 JAMES      CLERK
      7902 FORD       ANALYST
      7934 MILLER     CLERK
已选择 10 行。
```

此时您可以使用 TRUNCATE TABLE 语句来删除 staff1 表中所有的行，如例 10-45 所示。

例 10-45

```
SQL> TRUNCATE TABLE staff1;
```

例 10-45 结果

```
表已截掉。
```

现在再使用与例 10-44 完全相同的查询语句，您会得不到任何结果，如例 10-46 所示。

例 10-46

```
SQL> SELECT *
    2  FROM staff1;
```

例 10-46 结果

```
未选定行。
```

但是如果您使用 SQL*Plus 的 DESC 命令，会发现 staff1 表的结构依然存在，如例 10-47 所示。

例 10-47

```
SQL> DESC staff1;
```

例 10-47 结果

```
 名称                                      是否为空? 类型
 ----------------------------------------- ------ ------
 EMPNO                                            NUMBER(4)
 NAME                                             VARCHAR2(10)
 JOB                                              VARCHAR2(9)
```

此时如果您使用 DROP TABLE 语句来删除 staff1 表，staff1 表的结构还会存在吗？请看例 10-48 和例 10-49。

例 10-48

```
SQL> DROP TABLE staff1;
```

例 10-48 结果

```
表已丢弃。
```

例 10-49

```
SQL> DESC staff1
```

例 10-49 结果

```
ERROR:
ORA-04043: 对象 staff1 不存在
```

从例 10-49 的结果可以看出，staff1 表的结构已不存在了。

为了进一步演示 TRUNCATE TABLE 和 DROP TABLE 语句之间的差别，我们再举一个使用 DROP TABLE 语句的例子。现在您使用 DROP TABLE 语句来删除 staff 表，如例 10-50 所示。

例 10-50

```
SQL> DROP TABLE staff;
```

例 10-50 结果

```
表已丢弃。
```

您现在如果使用例 10-51 所示的查询语句，会得到表或视图不存在的出错信息。

例 10-51

```
SQL> SELECT *
2  FROM staff;
```

例 10-51 结果

```
FROM staff
     *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

如果您使用 SQL*Plus 的 DESC 命令，会发现 staff 表的结构也不存在了，如例 10-52 所示。

例 10-52

```
SQL> DESC staff
```

例 10-52 结果

```
ERROR:
ORA-04043: 对象 staff 不存在
```

相信通过以上的例子现在您可能已经理解了 TRUNCATE TABLE 和 DROP TABLE 语句的功能和它们之间的差别。如果您还是不理解的话，请您回忆一下三峡工程的壮观景象，您可以把 TRUNCATE TABLE 语句想象成三峡移民时人去楼空的景象，虽然人和东西都搬

走了，但建筑依然存在。而 DROP TABLE 语句就如同人工爆破，随着一声声巨响，那些废弃的建筑都夷为了平地。

10.10 小　　结

本章详细地介绍了包括创建表和删除表等有关维护表的语句，这些语句均为 DDL 语句，它们中的大多数是非常危险的语句，特别是 TRUNCATE TABLE 和 DROP TABLE 语句，因此建议在使用这两个语句之前最好做备份。

如果您曾读过其他类似的书，可能会发现本书这一部分的内容似乎比其他同类书籍的内容多。这是因为我个人认为表是数据库中最重要的对象，因为只有表中存有数据。所谓的数据库设计就是表的设计，它包含了表的结构（列和列数据类型等）的设计以及表与表之间的关系（主键和外键）设计。另外，数据库中的备份和恢复也是针对表的，因为在数据库中真正需要保护的是数据，而只有表中存储着数据。

下面我们用一个实际的例子来结束本章的讨论。这个例子是我为一个欧洲跨国公司做现场培训时遇到的。该公司有一个基于 Oracle 数据库的人事管理方面的软件。在该数据库中有两个表，一个表存放了所有员工的数据，但不包括员工和经理的关系；另一个表存放了所有员工和经理的关系和一些其他的附加信息。现在公司的一名高级经理要一份包括所有员工和他们经理的信息的报告。

为了模拟以上实例，我们先创建一个叫 e_m_shell 的表，它只包含两列 e_id 和 m_id，它们都为 4 位整数，e_id 为 primary key，如例 10-53 所示。

例 10-53

```
SQL> create table e_m_shell
  2          (e_id    number(4)  primary key,
  3           m_id    number(4));
```

例 10-53 结果

```
表已创建。
```

现在我们来检查一下 e_m_shell 表是否已建好及它的结构是否正确，如例 10-54 所示。

例 10-54

```
SQL> desc e_m_shell
```

例 10-54 结果

```
名称                                        是否为空? 类型
------------------------------------ -------- ---------
E_ID                                         NOT NULL NUMBER(4)
M_ID                                                  NUMBER(4)
```

例 10-54 结果告诉我们所创建的 e_m_shell 表准确无误。

之后我们把 emp 表中的 empno、mgr 数据插入到 e_m_shell 表中，如例 10-55 所示。

例 10-55

```
SQL> INSERT INTO e_m_shell(e_id, m_id)
  2            SELECT empno, mgr
  3            FROM   emp;
```

例 10-55 结果

```
已创建 14 行。
```

向 e_m_shell 表中插入了数据之后，我们用例 10-56 的查询语句来验证所插入的数据是否正确。

例 10-56

```
SQL> select * from e_m_shell;
```

例 10-56 结果

```
      E_ID       M_ID
---------- ----------
      7369       7902
      7499       7698
      7521       7698
      7566       7839
      7654       7698
      7698       7839
      7782       7839
      7788       7566
      7839
      7844       7698
      7876       7788
      7900       7698
      7902       7566
      7934       7782
已选择 14 行。
```

例 10-56 显示的结果正是我们所期望的。

建完了 e_m_shell 表，我们再建另一个表，表名为 emp_shell，它的数据与 emp 表一样。因此我们使用例 10-57 的 DDL 语句来创建 emp_shell 表。

例 10-57

```
SQL> create table emp_shell
  2  as select * from emp;
```

例 10-57 结果

```
表已创建。
```

为了模拟存放了所有员工的数据的表中没有员工和经理的关系这一事实，我们使用例 10-58 的 DDL 语句将 emp_shell 表中的 mgr 列删除掉。

例 10-58

```
SQL> alter table emp_shell
  2  drop column mgr;
```

例 10-58 结果

```
表已更改。
```

现在我们用例 10-59 的查询语句来验证所创建的 emp_shell 表是否正确。

例 10-59

```
SQL>  SELECT *
  2   FROM emp_shell;
```

例 10-59 结果

EMPNO	ENAME	JOB	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	17-12 月-80	800		20
7499	ALLEN	SALESMAN	20-2 月 -81	1600	300	30
7521	WARD	SALESMAN	22-2 月 -81	1250	500	30
7566	JONES	MANAGER	02-4 月 -81	2975		20
7654	MARTIN	SALESMAN	28-9 月 -81	1250	1400	30
7698	BLAKE	MANAGER	01-5 月 -81	2850		30
7782	CLARK	MANAGER	09-6 月 -81	2450		10
7788	SCOTT	ANALYST	19-4 月 -87	3000		20
7839	KING	PRESIDENT	17-11 月-81	5000		10
7844	TURNER	SALESMAN	08-9 月 -81	1500	0	30
7876	ADAMS	CLERK	23-5 月 -87	1100		20
7900	JAMES	CLERK	03-12 月-81	950		30
7902	FORD	ANALYST	03-12 月-81	3000		20
7934	MILLER	CLERK	23-1 月 -82	1300		10

已选择 14 行。

例 10-59 的结果告诉我们所创建的 emp_shell 表准确无误。

最后，利用 e_shell 和 e_m_shell 表写一个查询语句来显示员工和该员工经理的工号、名字和职位，如例 10-60 所示。

例 10-60

```
SQL> SELECT e.empno, e.ename, e.job, m.empno, m.ename, m.job
  2  FROM emp_shell e, e_m_shell, (SELECT empno, ename, job
  3                                FROM   emp) m
  4  WHERE e.empno = e_m_shell.e_id
  5  AND   e_m_shell.m_id = m.empno;
```

例 10-60 结果

```
     EMPNO ENAME      JOB              EMPNO ENAME      JOB
---------- ---------- --------- ---------- ---------- -------
      7369 SMITH      CLERK            7902 FORD       ANALYST
      7499 ALLEN      SALESMAN         7698 BLAKE      MANAGER
      7521 WARD       SALESMAN         7698 BLAKE      MANAGER
      7566 JONES      MANAGER          7839 KING       PRESIDENT
      7654 MARTIN     SALESMAN         7698 BLAKE      MANAGER
      7698 BLAKE      MANAGER          7839 KING       PRESIDENT
      7782 CLARK      MANAGER          7839 KING       PRESIDENT
      7788 SCOTT      ANALYST          7566 JONES      MANAGER
      7844 TURNER     SALESMAN         7698 BLAKE      MANAGER
      7876 ADAMS      CLERK            7788 SCOTT      ANALYST
      7900 JAMES      CLERK            7698 BLAKE      MANAGER
      7902 FORD       ANALYST          7566 JONES      MANAGER
      7934 MILLER     CLERK            7782 CLARK      MANAGER
已选择 13 行。
```

例 10-60 结果就是那位公司的高级经理所要的那份包括了所有员工和经理的信息的报告（在实际工作中，您应该为某些列定义别名，这样您的经理才容易理解）。

以上的实例除演示了如何解决实际问题外，还复习了前面讲过的一些重要内容。

细心的读者可以发现本章包含了 INSERT 语句，这是一个 DML 语句。不少同类的书是先讲 DML 语句，后讲 DDL 语句，但我觉得从逻辑顺序上来讲，应该是先创建一个表，然后才能对这个表进行 DML 操作。这个问题就有点像在上大学时，学数学分析与普通物理时所遇到的两难问题一样。在学数学分析时用到了普通物理，在学普通物理时又用到了数学分析，哪门课应该先学谁也说不清楚。

10.11　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 数据定义语言（DDL）。
- 如何创建表？
- CREATE TABLE 语句的用法。
- 表和列的命名和引用规则。
- 4 种常用的数据类型和默认值。
- 用子查询来创建表。
- 用子查询来创建表时要注意的事项。
- 如何在一个表中加入一列？
- 如何修改在一个表中已经存在的列？

- 如何从一个表中删除一列？
- 如何在一个表中把某一列设置成无用（UNUSED）？
- 如何删除已经设置为无用（UNUSED）的列？
- 把某一列设置成无用（UNUSED）时要注意的事项。
- 什么是对象？
- 如何修改对象的名字？
- 如何为表或列添加注释？
- 表的截断和删除。
- TRUNCATE TABLE 语句的特性。
- DROP TABLE 语句的特性。

第11章

替代变量

以前的各章中所讨论的 SQL 语句都有一个共同的特点，即每次执行同一 SQL 语句都会产生完全相同的结果（如果所操作的表没有改变过），这样的 SQL 语句在商业系统中几乎没什么实际的用处。SQL*Plus 的替代变量的引入就可以解决上面所说的问题。

11.1 替代变量引入的原因

您还记得在第 2 章中刚开始的例子吗？老板让您打印一份工资在 1500 元（包括 1500 元）或高于 1500 元的员工的清单，该例子与下面的例 11-1 相同。

例 11-1

```
SQL> SELECT empno, ename, sal
  2  FROM emp
  3  WHERE sal >= 1500;
```

例 11-1 结果

```
     EMPNO ENAME                       SAL
---------- -------------------- ----------
      7499 ALLEN                      1600
      7566 JONES                      2975
      7698 BLAKE                      2850
      7782 CLARK                      2450
      7788 SCOTT                      3000
      7839 KING                       5000
      7844 TURNER                     1500
      7902 FORD                       3000
已选择 8 行。
```

现在老板改主意了，他要您打印一份工资在 1600 元（包括 1600 元）或高于 1600 元的员工的清单。您无法重用以上的 SQL 语句，就因为相差了一个数字您就不得不重新输入与上面已经调试好的 SQL 语句几乎完全相同的语句。如果您的老板一天改几次主意，您又该怎么办呢？

11.2　以&开始的替代变量

请不用着急，Oracle 早就想到了这一点，它引入了替代变量，可以帮助您解决所遇到的难题。现在您可以使用例 11-2 的查询语句。

例 11-2

```
SQL> SELECT empno, ename, sal
    2  FROM emp
    3  WHERE sal >= &v_salary;
```

例 11-2 中的&v_salary 就是替代变量。变量的名字是您起的，只要您认为有意义就行，但别忘了在变量的名字之前加上一个&号。

当运行例 11-2 的 SQL 语句时，Oracle 系统会给出“输入 v_salary 的值:”的提示，此时可以输入任何您感兴趣的工资值，如 1600，按 Enter 键之后，您就会得到所需要的结果。

例 11-2 系统提示和输入

输入 v_salary 的值：1600

注意：

上面的阴影部分为系统提示，余下的部分为输入的值。

例 11-2 结果

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7499 ALLEN            1600
      7566 JONES            2975
      7698 BLAKE            2850
      7782 CLARK            2450
      7788 SCOTT            3000
      7839 KING             5000
      7902 FORD             3000
已选择 7 行。
```

现在您可以使用例 11-3 的 SQL*Plus 命令将例 11-2 的 SQL 语句存在一个名为 sal 的脚本文件中，之后您就可以反复地使用这个脚本文件了。

例 11-3

```
SQL> save d:\sql\sal replace
```

例 11-3 结果

```
已写入文件 d:\sql\sal.sql
```

如果老板又改主意了，他要您打印一份工资在 2000 元或高于 2000 元的员工的清单。您只要重新运行脚本文件 sal，并在 Oracle 系统给出“输入 v_salary 的值:”的提示时输入

2000 即可，如例 11-4 所示。

例 11-4

```
SQL> @d:\sql\sal
```

例 11-4 系统提示和输入

输入 v_salary 的值：2000

☠ **注意：**

上面的阴影部分为系统提示，余下的部分为您输入的值。

例 11-4 结果

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7566 JONES            2975
      7698 BLAKE            2850
      7782 CLARK            2450
      7788 SCOTT            3000
      7839 KING             5000
      7902 FORD             3000
已选择 6 行。
```

例 11-4 的结果正是您的老板所要的。现在不管老板如何“变”，您都可以用这个脚本文件来 “以不变应万变”了。

如果您想知道 sal 的原值和新值，可以先输入例 11-5 的 SQL*Plus 命令。

例 11-5

```
SQL> SET VERIFY ON
```

之后再使用例 11-6 的 SQL*Plus 命令运行脚本文件 sal。

例 11-6

```
SQL> @d:\sql\sal
```

例 11-6 系统提示和输入

输入 v_salary 的值：1700

原值　3: WHERE sal >= &v_salary

新值　3: WHERE sal >= 1700

☠ **注意：**

上面的阴影部分为系统提示，余下的部分为您输入的值。

例 11-6 结果

```
     EMPNO ENAME             SAL
---------- ---------- ----------
```

```
      7566 JONES              2975
      7698 BLAKE              2850
      7782 CLARK              2450
      7788 SCOTT              3000
      7839 KING               5000
      7902 FORD               3000
已选择 6 行。
```

一般只是在调试阶段才可能将 VERIFY 设置成 ON。当 SQL 语句是嵌在程序或脚本文件中时，而该程序或脚本文件又是为用户开发的，在它们交付使用之前一定要将 VERIFY 置成 OFF。因为用户看到那些莫名其妙的显示时会感到困惑。

11.3　字符型和日期型替代变量

在解释字符型和日期型替代变量之前，我们先看一个例子，如例 11-7 所示。

例 11-7

```
SQL> SELECT empno, ename, sal, job, deptno
   2  FROM emp
3  WHERE job = &v_job;
```

例 11-7 系统提示和输入

输入 v_job 的值：CLERK

例 11-7 结果

```
原值   3: WHERE job = &v_job
新值   3: WHERE job = CLERK
WHERE job = CLERK
          *
ERROR 位于第 3 行:
ORA-00904: 无效列名
```

当运行例 11-7 的 SQL 语句时，在系统提示输入时您输入了 CLERK，但系统却返回了出错信息。这是为什么呢？

这是因为 Oracle 系统在处理 SQL*Plus 替代变量时，只是把替代变量（&v_job）简单地替换成我们在系统提示下输入的值（CLERK）。因为 job 这一列的数据类型为字符型，所以 CLERK 必须用单引号括起来。为了以后的演示方便，您可以先将例 11-7 的 SQL 语句存入一个名为 JOB 的脚本文件中，如例 11-8 所示。

例 11-8

```
SQL> SAVE D:\SQL\JOB
```

例 11-8 结果

```
已创建文件 D:\SQL\JOB.sql
```

为了以后的显示容易理解，应该将 VERIFY 设置成 OFF，如例 11-9 所示。

例 11-9

```
SQL> set verify off
```

现在使用例 11-10 的 SQL*Plus 命令运行脚本文件 JOB。

例 11-10

```
SQL>  @D:\SQL\JOB
```

之后在系统提示输入时输入'CLERK'。

例 11-10 系统提示和输入

输入 v_job 的值：'CLERK'

例 11-10 结果

```
     EMPNO  ENAME                 SAL  JOB              DEPTNO
----------  ----------  ----------  ---------  --------
      7369  SMITH                 800  CLERK                20
      7876  ADAMS                1100  CLERK                20
      7900  JAMES                 950  CLERK                30
      7934  MILLER               1300  CLERK                10
```

这次您终于得到了预期的结果。

注意：

值得注意的是，当 SQL*Plus 进行变量替换时除了数据类型之外，对用户的输入不做任何的检查。

我们使用例 11-11 的 SQL*Plus 命令重新运行脚本文件 JOB 来演示一种难以理解但可能出现的输入错误。

例 11-11

```
SQL> @D:\SQL\JOB
```

之后在系统提示输入时输入'CLERK';。

例 11-11 系统提示和输入

输入 v_job 的值：'CLERK';

例 11-11 结果

```
WHERE job = 'CLERK';
                   *
ERROR 位于第 3 行:
ORA-00911: 无效字符
```

由于我们习惯地在输入的结尾加上了分号（;），Oracle 系统返回了出错信息。

一般只有一个 SQL 语句需要反复执行时才使用替代变量，这样的 SQL 语句多数都是

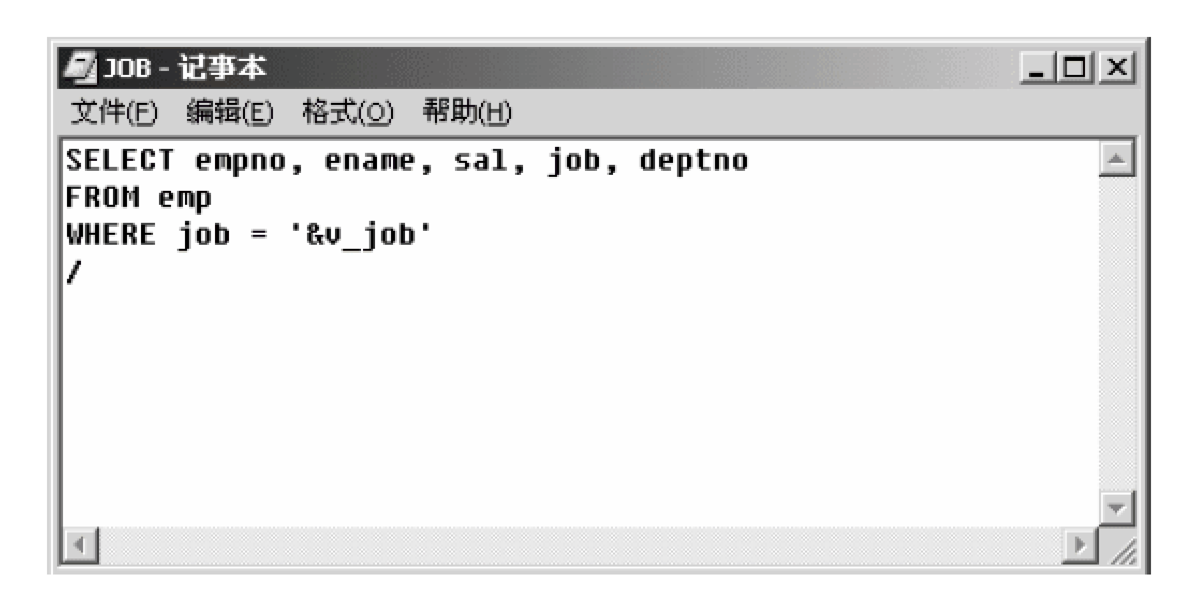
嵌在程序或脚本文件中的，而这样的程序或脚本文件又是为用户开发的，您很难知道用户对 SQL 语句和 SQL*Plus 命令的掌握程度。因此，作为软件的开发者，您应该保证使提示信息一目了然，而且用户要输入的信息越简单且越少越好。所以最好的办法是应该把 SQL 语句中的字符型和日期型替代变量用单引号括起来。

现在我们利用例 11-12 的 SQL*Plus 命令来编辑脚本文件 JOB。

例 11-12

```
SQL> EDIT D:\SQL\JOB
```

例 11-12 结果



现在在编辑器中把替代变量&v_job 用单引号括起来。然后选择“文件”→“保存”命令保存，最后退出编辑器。

再使用例 11-13 的 SQL*Plus 命令来运行脚本文件 JOB。

例 11-13

```
SQL> @D:\SQL\JOB
```

之后在系统提示输入时输入 CLERK。

例 11-13 系统提示和输入

输入 v_job 的值：CLERK

例 11-13 结果

```
     EMPNO ENAME             SAL JOB          DEPTNO
---------- ---------- ---------- --------- ---------
      7369 SMITH             800 CLERK            20
      7876 ADAMS            1100 CLERK            20
      7900 JAMES             950 CLERK            30
      7934 MILLER           1300 CLERK            10
```

这次虽然只输入了 CLERK，但却得到了预期的结果。

11.4　以&&开始的替代变量

在介绍完了以 & 开始的替代变量之后，我们要介绍另一类替代变量，即以 && 开始的替代变量。为了说明它们之间的区别，请先看例 11-14。

例 11-14

```
SQL> SELECT ename, job, &&v_col
   2  FROM emp
3  ORDER BY &v_col;
```

之后在系统提示输入时输入 sal。

例 11-14 系统提示和输入

输入 v_col 的值：sal

例 11-14 结果

```
ENAME                JOB              SAL
---------- --------- ----------
SMITH                CLERK             800
JAMES                CLERK             950
ADAMS                CLERK            1100
WARD                 SALESMAN         1250
MARTIN               SALESMAN         1250
MILLER               CLERK            1300
TURNER               SALESMAN         1500
ALLEN                SALESMAN         1600
CLARK                MANAGER          2450
BLAKE                MANAGER          2850
JONES                MANAGER          2975
SCOTT                ANALYST          3000
FORD                 ANALYST          3000
KING                 PRESIDENT        5000
已选择 14 行。
```

从例 11-14 可以看出，尽管在该例的 SQL 语句中替代变量 v_col 出现了两次，但 Oracle 系统只提示输入一次。

如果您想重用某个替代变量而不想让 Oracle 系统每次提示输入，可以使用以&&开始的替代变量，这样 Oracle 系统只在第 1 次使用这个替代变量时提示您输入它，以后再使用该替代变量时，Oracle 系统就不再提示输入，而是使用您第 1 次输入的值。

那么如果重新输入例 11-14 的 SQL 语句会得到什么样的结果呢？您可以自行尝试。

其实，Oracle 系统将没有任何提示而是直接产生和例 11-14 显示的结果完全相同的输出，这是因为 Oracle 系统已经记住了您第 1 次输入的值。

如果想改变这一替代变量的值，例如，这次您想知道的不是工资(sal)而是佣金(comm)，可以使用 SQL*Plus 的 UNDEFINE 命令来取消这个变量，如例 11-15 所示。

例 11-15

```
SQL> undefine v_col
```

现在您可以输入与例 11-14 完全相同的 SQL 语句，如例 11-16 所示。

例 11-16

```
SQL> SELECT ename, job, &&v_col
  2  FROM emp
  3  ORDER BY &v_col;
```

之后在系统提示输入时输入 comm。

例 11-16 系统提示和输入

输入 v_col 的值：comm

例 11-16 结果

```
ENAME      JOB             COMM
---------- --------- ----------
TURNER     SALESMAN           0
ALLEN      SALESMAN         300
WARD       SALESMAN         500
MARTIN     SALESMAN        1400
SMITH      CLERK
JONES      MANAGER
JAMES      CLERK
MILLER     CLERK
FORD       ANALYST
ADAMS      CLERK
BLAKE      MANAGER
CLARK      MANAGER
SCOTT      ANALYST
KING       PRESIDENT
已选择 14 行。
```

例 11-16 的显示结果正是您所期望的。

除了以上介绍的使用替代变量的两种方法之外，您还可以使用 SQL*Plus 命令 DEFINE 和 ACCEPT 来定义替代变量。我们将在本章后面介绍这两个命令。下面先介绍替代变量在 SQL 语句中可以出现的地方。

11.5　替代变量可以出现的地方

替代变量几乎可以出现在 SQL 语句中的任何地方，例如：

- SELECT 子句中。
- ORDER BY 子句中。
- WHERE 子句中（替代变量甚至可以替代整个的条件表达式）。
- 任何可以使用列或表达式的地方。
- 表名出现的地方。

● 整个的查询语句。

下面我们用一个比较全面的例子来演示上面所说的替代变量的使用，如例 11-17 所示。

例 11-17

```
SQL> SELECT &column1, &column2
  2  FROM &table_name
  3  WHERE &condition
  4  ORDER BY &sorting;
```

运行以上的 SQL 语句之后，在系统提示输入时输入相关的替代变量的值。

例 11-17 系统提示和输入

输入 column1 的值：dname

输入 column2 的值：loc

输入 table_name 的值：dept

输入 condition 的值：deptno <> 10

输入 sorting 的值：dname

例 11-17 结果

```
DNAME          LOC
-------------- -------
OPERATIONS      BOSTON
RESEARCH       DALLAS
SALES          CHICAGO
```

在例 11-17 中利用替代变量得到了有关 dept 表的信息。您可以再次运行相同的 SQL 语句，之后在系统提示输入时输入不同的替代变量的值，就会得到不同的结果。例如，输入有关 emp 表的信息，如例 11-18 所示。

例 11-18

```
SQL> SELECT &column1, &column2
  2  FROM &table_name
  3  WHERE &condition
  4  ORDER BY &sorting;
```

运行以上的 SQL 语句，之后在系统提示输入时输入相关的替代变量的值。

例 11-18 系统提示和输入

输入 column1 的值：ename

输入 column2 的值：sal

输入 table_name 的值：emp

输入 condition 的值：job = 'CLERK'

输入 sorting 的值：sal

例 11-18 结果

```
ENAME             SAL
---------- ----------
SMITH             800
JAMES             950
ADAMS            1100
MILLER           1300
```

从例 11-18 的显示结果可以看出，这次得到的是 emp 表中的信息。

11.6　使用DEFINE定义替代变量

除了以上介绍的使用替代变量的两种方法之外，还可以使用 SQL*Plus 命令 DEFINE 来定义替代变量。需要注意的是，DEFINE 命令只能用来创建字符型的替代变量。下面的例 11-19 就是一个创建字符型替代变量的例子。

例 11-19

```
SQL> DEFINE v_job = CLERK
```

现在可以输入例 11-20 的 SQL 语句。

例 11-20

```
SQL> SELECT ename, sal, job
   2  FROM emp
   3  WHERE job = '&v_job'
   4  ORDER BY sal;
```

由于我们已经使用例 11-19 的 SQL 语句将替代变量 v_job 赋为 CLERK，所以 Oracle 系统已经记住了这次的输入值，并不再提示输入，而是直接把 CLERK 作为变量 v_job 的值来使用。因此，当输入例 11-20 的 SQL 语句后，Oracle 系统将没有任何提示而直接产生例 11-20 的输出结果。

例 11-20 结果

```
ENAME             SAL JOB
---------- ---------- -----
SMITH             800 CLERK
JAMES             950 CLERK
ADAMS            1100 CLERK
MILLER           1300 CLERK
```

可以使用以下的 SQL*Plus 命令来创建一个字符型的替代变量，并为它赋值（如例 11-19）：

DEFINE 替代变量名=变量值

可以使用以下的 SQL*Plus 命令显示一个替代变量、替代变量的值及其数据类型（如例 11-21）：

DEFINE 替代变量名

例 11-21

```
SQL> DEFINE v_job
```

例 11-21 结果

```
DEFINE V_JOB           = "CLERK" (CHAR)
```

也可以使用以下的 SQL*Plus 命令显示所有的替代变量、替代变量值及其数据类型（如例 11-22）：

DEFINE

例 11-22

```
SQL> DEFINE
```

例 11-22 结果

```
DEFINE _SQLPLUS_RELEASE = "900010001" (CHAR)
DEFINE _EDITOR         = "Notepad" (CHAR)
With the Partitioning option
JServer Release 9.0.1.1.1 - Production" (CHAR)
DEFINE _O_RELEASE      = "900010301" (CHAR)
DEFINE V_JOB           = "CLERK" (CHAR)
```

虽然您已经学习了几种定义替代变量的方法，但所有这些方法都有一个共同的缺陷，就是系统产生的提示信息非常不清楚。用这样的方法开发的软件用户是很难满意的。不过，Oracle 又一次“高瞻远瞩”地想到了这一点。为此 Oracle 引入了 ACCEPT 命令，用该命令来定义替代变量就可以帮助您解决所遇到的难题。

11.7　使用ACCEPT定义替代变量

SQL*Plus 的 ACCEPT 命令在开发软件或脚本文件时很有用，该命令的基本功能是读入用户的输入并把它存入到一个变量中。当在接收用户的输入时，可以用该命令建立个性化的输入提示信息。我们利用例 11-23~例 11-32 来演示这一命令的用法。

例 11-23

```
SQL> SELECT ename, sal, job
  2  FROM emp
  3  WHERE job = UPPER('&v_job')
  4  ORDER BY sal;
```

运行以上的 SQL 语句之后在系统提示输入时，输入替代变量的值 clerk。

例 11-23 系统提示和输入

输入 v_job 的值：clerk

之后系统会产生如下的输出。

例 11-23 结果

```
ENAME             SAL  JOB
---------- ---------- -----
SMITH             800  CLERK
JAMES             950  CLERK
ADAMS            1100  CLERK
MILLER           1300  CLERK
```

现在可以使用例 11-24 的 SQL*Plus 命令将例 11-23 的 SQL 语句存在名为 acc_job 的脚本文件中。

例 11-24

```
SQL> save d:\sql\acc_job
```

例 11-24 结果

```
已创建文件 d:\sql\acc_job.sql
```

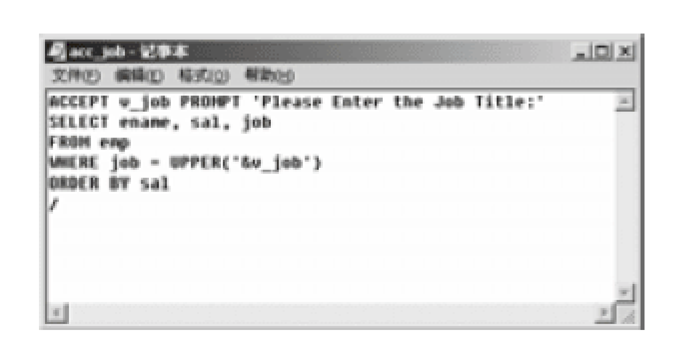
之后可以用例 11-25 的 SQL*Plus 命令来编辑刚创建的脚本文件 acc_job。

例 11-25

```
SQL> edit d:\sql\acc_job
```

当编辑窗口出现时，要在第 1 行加入“ACCEPT v_job PROMPT 'Please Enter the Job Title:'”，之后选择“文件”→“保存”命令，最后退出编辑器，如例 11-26 所示。

例 11-26



现在可以使用例 11-27 的 SQL*Plus 命令来运行脚本文件 acc_job。

例 11-27

```
SQL> @d:\sql\acc_job
```

运行以上的 SQL*Plus 命令之后，系统会产生如下所示的用户友好的提示信息。相信这样的提示信息用户很容易理解，也就不容易产生输入错误。

例 11-27 系统提示和输入

请您输入职位（Job）：clerk

例 11-27 结果

```
ENAME             SAL  JOB
---------- ---------- -----
```

```
SMITH              800  CLERK
JAMES              950  CLERK
ADAMS             1100  CLERK
MILLER            1300  CLERK
```

如果您的操作系统和数据库字的字符集都为中文，则可以使用如下的方法将系统提示信息改为中文。首先使用例 11-28 的 SQL*Plus 命令将脚本文件 acc_job.sql 装入 SQL 缓冲区。

例 11-28

```
SQL> get d:\sql\acc_job
```

例 11-28 结果

```
  1  ACCEPT v_job  PROMPT 'Please Enter the Job Title:'
  2  SELECT ename, sal, job
  3  FROM emp  4  WHERE job = UPPER('&v_job')
  5* ORDER BY sal
```

现在可以使用例 11-29 的 SQL*Plus 命令将 SQL 缓冲区中的 SQL 语句存入名为 chacc_job 的脚本文件中。

例 11-29

```
SQL> save d:\sql\chacc_job replace
```

例 11-29 结果

```
已写入文件 d:\sql\chacc_job.sql
```

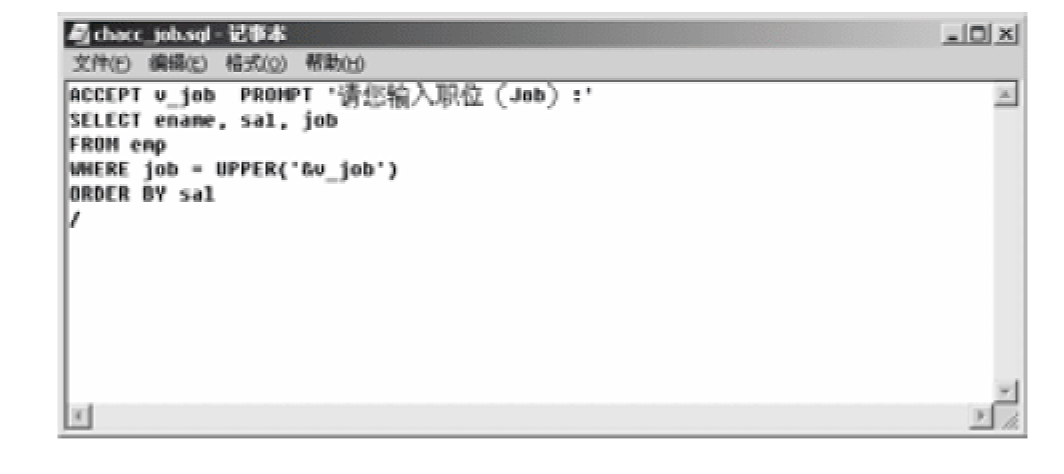
之后可以用例 11-30 的 SQL*Plus 命令来编辑刚创建的脚本文件 chacc_job.sql。

例 11-30

```
SQL> edit d:\sql\chacc_job.sql
```

当编辑窗口出现时，要在第 1 行加入“ACCEPT v_job PROMPT '请您输入职位(Job):'”，之后选择“文件”→“保存”命令，最后退出编辑器，如例 11-31 所示。

例 11-31



现在可以使用例 11-32 的 SQL*Plus 命令来运行脚本文件 chacc_job.sql。

例 11-32

```
SQL> @d:\sql\chacc_job.sql
```

运行以上的 SQL*Plus 命令之后，系统会产生例 11-32 所示用户友好的中文提示信息。

相信这样的提示信息用户很容易理解，也就不容易产生输入错误了。

例 11-32 系统提示和输入

请您输入职位（Job）：clerk

例 11-32 结果

```
ENAME              SAL  JOB
---------- ---------- -----
SMITH              800  CLERK
JAMES              950  CLERK
ADAMS             1100  CLERK
MILLER            1300  CLERK
```

11.8 如何使用ACCEPT命令的HIDE选项

如果公司中的某些信息只有经理们可以看，可以使用 ACCEPT 命令的 HIDE 选项。该选项将使 ACCEPT 命令不显示用户的输入，和在操作系统下输入口令类似。

下面我们用一个简单的例子来演示如何开发一个使用该功能的脚本文件。可以输入如例 11-33 的查询语句，该查询语句只有当用户在系统提示输入 MANAGER 时，系统才会显示 emp 表中的雇员名（ename）、工资（sal）和职位（job）。

例 11-33

```
SQL> SELECT ename, sal, job
  2  FROM emp
  3  WHERE '&pwd' = 'MANAGER';
```

运行以上的 SQL 语句之后和系统提示输入时，要输入替代变量的值 MANAGER。

例 11-33 系统提示和输入

输入 pwd 的值：MANAGER

之后系统会输出每个员工的名字（ename）、工资（sal）和职位（job）。

例 11-33 结果

```
ENAME              SAL  JOB
---------- ---------- ---------
SMITH              800  CLERK
ALLEN             1600  SALESMAN
WARD              1250  SALESMAN
JONES             2975  MANAGER
MARTIN            1250  SALESMAN
BLAKE             2850  MANAGER
CLARK             2450  MANAGER
SCOTT             3000  ANALYST
KING              5000  PRESIDENT
```

```
TURNER              1500  SALESMAN
ADAMS               1100  CLERK
JAMES                950  CLERK
FORD                3000  ANALYST
MILLER              1300  CLERK
已选择 14 行。
```

现在可以使用如例 11-34 的 SQL*Plus 命令将刚测试好的 SQL 语句存入 D:\SQL 目录下（文件夹）的 PWD 脚本文件中。

例 11-34

```
SQL> SAVE D:\SQL\PWD
```

之后运行如例 11-35 的 SQL*Plus 命令来编辑刚存入 PWD 脚本文件中的 SQL 语句。

例 11-35

```
SQL> EDIT D:\SQL\PWD
```

在编辑器中的 SQL 语句之前（可以是第 1 行）输入 ACCEPT pwd PROMPT 'Please Enter Your Password:' HIDE，如例 11-36 所示。

例 11-36



保存刚才所做的变化，然后退出编辑器。

现在可以使用例 11-37 的 SQL*Plus 命令来运行刚测试好的 PWD 脚本文件中的 SQL 语句。

例 11-37

```
SQL> @d:\sql\pwd
```

运行以上的 PWD 脚本文件之后，在系统提示输入时输入替代变量的值 MANAGER。

例 11-37 系统提示和输入

请输入你的口令：*******

与例 11-33 不同的是，当您输入 MANAGER 时，系统不显示输入而是显示*******。这样就没人能偷看到口令（Password）。例 11-37 的显示输出与例 11-33 的完全相同。

例 11-37 结果

```
ENAME                SAL  JOB
---------- ---------- ---------
SMITH                800  CLERK
ALLEN               1600  SALESMAN
```

```
WARD                1250  SALESMAN
JONES               2975  MANAGER
MARTIN              1250  SALESMAN
BLAKE               2850  MANAGER
CLARK               2450  MANAGER
SCOTT               3000  ANALYST
KING                5000  PRESIDENT
TURNER              1500  SALESMAN
ADAMS               1100  CLERK
JAMES                950  CLERK
FORD                3000  ANALYST
MILLER              1300  CLERK
已选择 14 行。
```

如果在系统提示输入时输入的不是 MANAGER 或者不是大写的 MANAGER，系统会产生什么样的输出呢？请看例 11-38。

例 11-38

```
SQL> @D:\SQL\PWD
```

这次在系统提示输入时输入 manager。

例 11-38 系统提示和输入

请输入你的口令：*******

您会发现系统不输出任何员工的信息，如例 11-38 的结果。其实，只要输入的不是 MANAGER 就得不到任何员工的信息。

例 11-38 结果

```
未选定行
```

现在如果您去应征工作，当对方问您是否会写 Oracle 脚本文件或是否会用 Oracle 开发软件，您应该非常自信地回答“会”。

11.9　ACCEPT命令的格式和选项

除了上面所介绍的 ACCEPT 命令的用法之外，还可以使用该命令显式地定义日期型或数字型变量。ACCEPT 命令的格式如下：

ACCEPT　变量名　[数据类型] [FORMAT　格式化模式] [PROMPT　正文] [HIDE]

其中参数含义如下。

- 变量名：存储输入值的变量名。如果该变量不存在，SQL*Plus 会自动创建它。
- 数据类型：为数字型、日期型和字符型。字符型的最大长度是 240 个字节。
- FORMAT　格式化模式：它的定义与第 9 章中的一样，如 99,999.00 或 A28 等。
- PROMPT　正文：在用户可以输入变量的值之前系统的提示信息。

- HIDE：系统不显示用户的输入值，如口令（PASSWORD）。

另外值得注意的是，在 ACCEPT 关键字之后的替代变量之前不能加&符号，但该替代变量被引用时在其前面要加&符号。

11.10　参数和替代变量的永久设置

提示：

如果读者在阅读本节时，理解上有困难的话，请不要担心。即使不理解本节的内容也不会影响以后的学习。在本书的第 1 版并未包含这部分的内容，其实 Oracle 官方的 SQL 培训也没有包括这些内容，但了解以下的内容会使读者的日常工作和项目开发变得更简单、更方便。

假设现在您正在参加一个项目，您在每次启动 SQL*Plus 时都要进行同样的参数设置和替代变量的定义，这样做不但很麻烦而且也容易出错。能否将这些参数和变量定义一次，以后在项目进行中就一直使用下去呢？当然可以，这就需要使用 login.sql 脚本文件。这一文件由用户自己在工作目录（即启动 SQL*Plus 的命令）中创建，每次 SQL*Plus 启动时，首先读 login.sql 文件并使用其中的定义来设置参数和替代变量。以下就是具体操作步骤：

（1）打开记事本并在其中写入如下的参数设置和替代变量定义，如图 11.1 所示。

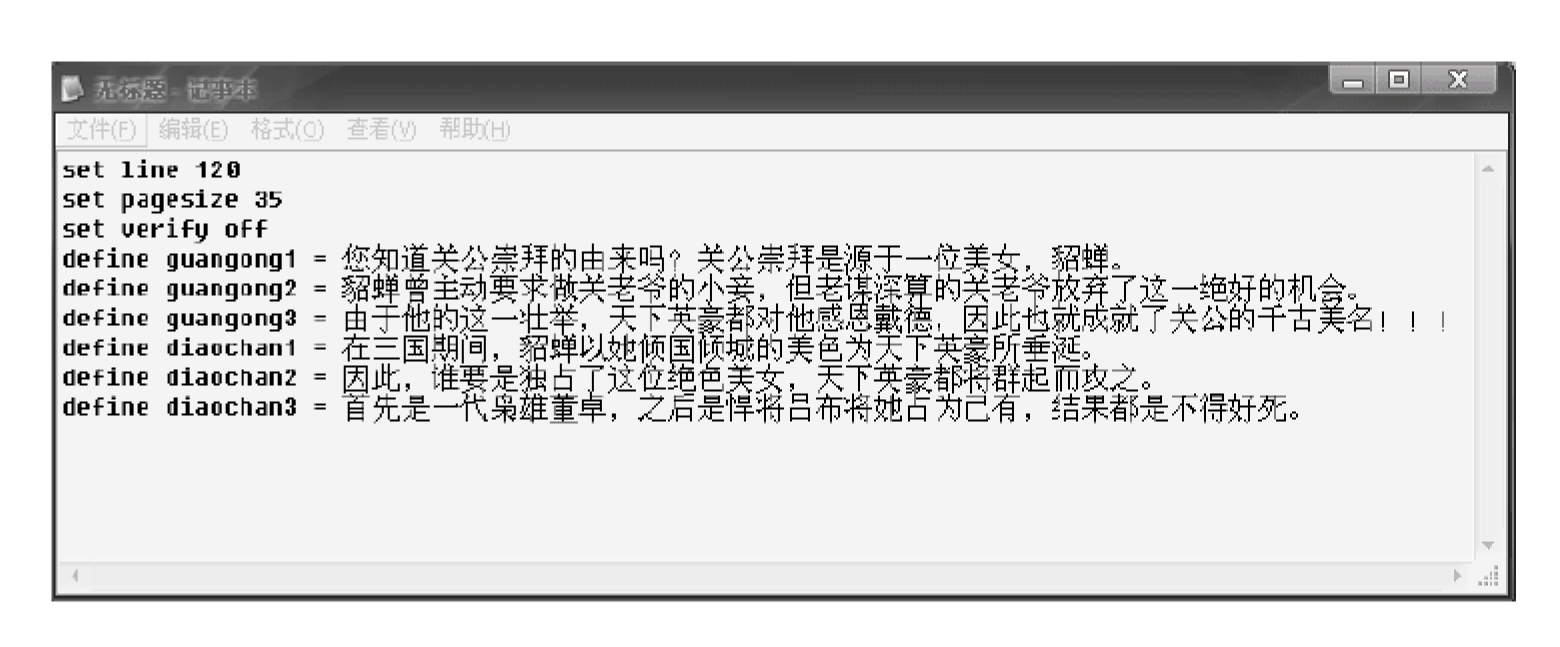


图　11.1

（2）选择“文件”→“保存”命令，弹出“另存为”对话框，在“保存在”下拉列表框中选择 E 盘的 SQL，在“文件名”下拉列表框中输入“login.sql”，最后单击“保存”按钮，如图 11.2 所示。

（3）进入 E 盘的 SQL 目录就可以发现刚生成的 login.sql 文件，如图 11.3 所示。

（4）启动 DOS 窗口，使用 e:命令切换到 E 盘，使用 cd sql 命令进入 E 盘的 SQL 目录。然后利用 sqlplus scott/tiger 命令启动 sqlplus 并以 scott 用户身份登录 Oracle 数据库，如图 11.4 所示。

（5）使用 sqlplus 的 show line、show pagesize 和 define 等命令可以验证所有的设置是否成功，如图 11.5 所示。



图　11.2



图　11.3

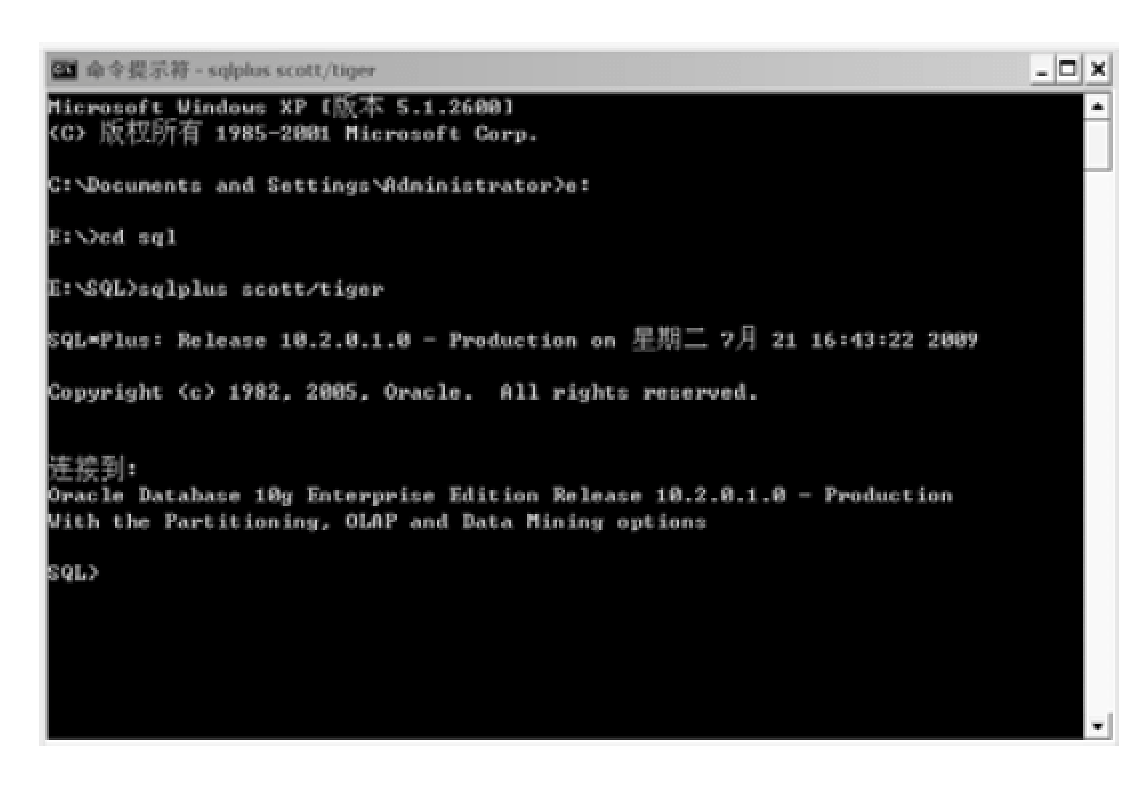


图　11.4

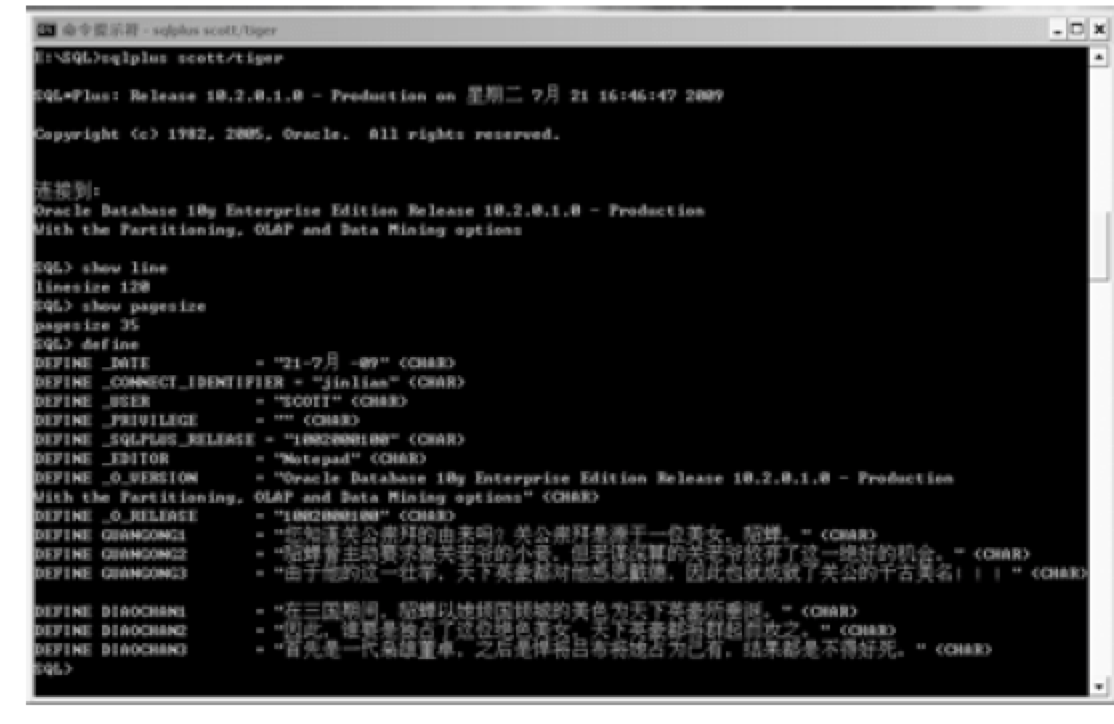


图　11.5

（6）启动 Oracle 公司称之为图形界面的 sqlplus 并以 scott 用户身份登录数据库，之后使用 sqlplus 的 show line、show pagesize 和 define 命令就会发现所有配置还是 Oracle 的默认值，如图 11.6 所示。这是因为 sqlplus 只在当前目录下寻找 login.sql 文件，而图形界面的 sqlplus 的当前目录并不是 E:\SQL。

（7）进入 E:\SQL 目录，复制该目录中的 login.sql 文件，如图 11.7 所示。

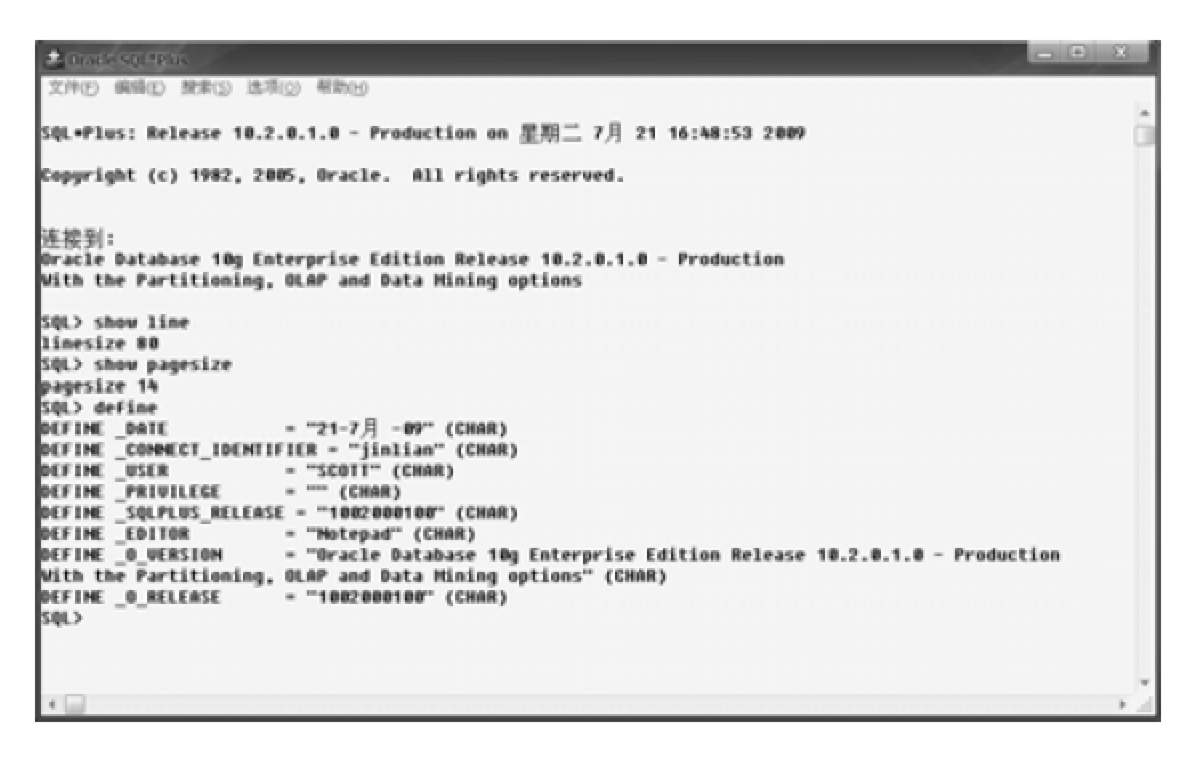


图　11.6



图　11.7

（8）进入图形界面 sqlplus 的当前目录 E:\oracle\product\10.2.0\db_1\BIN，将 login.sql

文件粘贴到该目录下，如图 11.8 所示。

（9）重新启动图形界面的 SQL*plus，之后使用 sqlplus 的 show line、show pagesize 和 define 等命令来验证所有的设置是否成功，如图 11.9 所示。这次您会发现所有的设置已经改变了。



图 11.8



图 11.9

最后可以使用例 11-39 的 SQL 查询语句来分行显示所定义的 SQL*Plus 替代变量的值，其中，chr 是一个函数，它将一个数字转换成 ASCII 码。chr(10)是将数字 10 转换成 ASCII 码，实际上与 10 对应的 ASCII 码是回车。这里需要使用连接符||，而不是逗号，“关公崇拜的由来”为整个字符串表达式的别名。

例 11-39

```
SQL> select '&guangong1'||chr(10)||
  2         '&diaochan1'||chr(10)||
  3         '&diaochan2'||chr(10)||
  4         '&diaochan3'||chr(10)||
  5         '&guangong2'||chr(10)||
  6         '&guangong3'||chr(10)  "关公崇拜的由来"
  7  from dual;
```

例 11-39 结果

```
关公崇拜的由来
----------------------------------------------------------------------
您知道关公崇拜的由来吗？关公崇拜是源于一位美女，貂蝉。
在三国期间，貂蝉以她倾国倾城的美色为天下英豪所垂涎。
因此，谁要是独占了这位绝色美女，天下英豪都将群起而攻之。
首先是一代枭雄董卓，之后是悍将吕布将她占为己有，结果都是不得好死。
貂蝉曾主动要求做关老爷的小妾，但老谋深算的关老爷放弃了这一绝好的机会。
由于他的这一壮举，天下英豪都对他感恩戴德，因此也就成就了关公的千古美名！！！
```

看来将好的东西、大家都喜欢的东西（是否也包括名誉、地位和财富？）据为己有是一件相当危险的事，很可能要付出生命的代价。

11.11　小　　结

本章详细地介绍了在利用 SQL 和 SQL*Plus 命令开发实用软件时可能会经常使用的 SQL*Plus 替代变量以及它们的不同定义和控制方法，其中，ACCEPT 命令功能最强。使用 ACCEPT 命令可以开发出非常个性化或用户友好的脚本文件。

尽管 SQL*Plus 替代变量为软件开发提供了很大的方便，但它们并没有为软件开发系上“安全带”。SQL*Plus 除了数据类型之外对用户的输入不做任何检查。因此，作为软件的开发者，您应该保证使提示信息一目了然，而且用户要输入的信息越简单且越少越好，所以，最好的办法是把 SQL 语句中的字符型和日期型替代变量用单引号括起来。

11.12　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- SQL*Plus 的替代变量。
- 以&开始的替代变量。
- SET VERIFY ON/OFF 的用法。
- 字符型和日期型替代变量的处理。
- 以&&开始的替代变量。
- 以&开始和以&&开始的替代变量之间的区别。
- 如何取消一个变量的定义？
- 替代变量可以出现的地方。
- DEFINE 命令的用法。
- ACCEPT 命令的用法。
- ACCEPT 命令的选项。
- 如何创建和编辑脚本文件？
- 如何建立个性化的输入提示信息？
- ACCEPT 命令中的 HIDE 选项的用法。
- DEFINE 命令和 ACCEPT 命令的区别。

第12章

数据的维护

数据的维护包括了数据操作语言 DML（Data Manipulating Language）和事务控制（Transaction Control）两大部分。有些专家认为数据操作语言是 SQL 语言的核心，因为只有 DML 操作才能改变数据库中的数据。

其实 DML 只包含了 INSERT、UPDATE 和 DELETE 3 个语句。

12.1 准备工作

为了数据库的安全及讲解方便，我们先使用如例 12-1 和例 12-3 的 SQL 语句创建两个临时的表 emp_DML 和 dept_DML。因为本章的大多数命令都很“危险”，所以在这一章中所有的命令将使用这两个临时的表。

例 12-1

```
SQL> CREATE TABLE emp_DML
  2  AS
  3  SELECT *
  4  FROM emp;
```

例 12-1 结果

```
表已创建。
```

当 SQL*Plus 显示表已创建之后，可以使用类似于例 12-2 的查询语句来检查刚创建的 emp_DML 表是否正确。

例 12-2

```
SQL> SELECT ename, job, sal
2  FROM emp_DML;
```

例 12-2 结果

```
结果已省略。
```

当创建了 emp_DML 表之后，就可以使用例 12-3 的 SQL 语句创建 dept_DML 表。

例 12-3

```
SQL>  CREATE TABLE dept_DML
  2   AS
```

```
3   SELECT *
4   FROM dept;
```

例 12-3 结果

```
表已创建。
```

当 SQL*Plus 显示表已创建之后，可以使用类似于例 12-4 的查询语句来检查一下刚创建的 dept_dml 表是否正确。

例 12-4

```
SQL> select * from dept_dml;
```

例 12-4 结果

```
    DEPTNO  DNAME           LOC
----------  --------------  --------
        10  ACCOUNTING      NEW YORK
        20  RESEARCH        DALLAS
        30  SALES           CHICAGO
        40  OPERATIONS      BOSTON
```

12.2　INSERT语句

当向一个表中添加一行新的数据时，需要使用 DML 语言中的 INSERT 语句。

该语句的格式如下：

INSERT INTO　表名　　[(列名 [, 列名…])]

VALUES　　　　　　　(数值 [, 数值…]);

其中各参数的含义如下。

- 表名：要输入数据的表的名字。
- 列名：表中要输入数据列的名字。
- 数值：对应列的具体值。

使用以上的 INSERT 语句格式每一次只能向表中插入一行数据。

例 12-5 的 DML 语句用于向 dept_dml 表中插入一行数据。

例 12-5

```
SQL> INSERT INTO dept_dml (deptno, dname, loc)
  2  VALUES             (66, '美容', '煤球胡同');
```

例 12-5 结果

```
已创建 1 行。
```

当向 dept_dml 表中插入一行数据之后，可以使用例 12-6 的查询语句来检查例 12-5 的 DML 语句是否正确。

例 12-6

```
SQL> SELECT *
  2  FROM dept_dml;
```

例 12-6 结果

```
 DEPTNO DNAME          LOC
---------- -------------- ---------
       10 ACCOUNTING      NEW YORK
       20 RESEARCH        DALLAS
       30 SALES           CHICAGO
       40 OPERATIONS      BOSTON
       66 美容            煤球胡同
```

例 12-6 的显示结果说明您的操作准确无误。

请注意在使用例 12-5 的 DML 语句插入一行数据时，日期型和字符型的数据必须用单引号括起来。您可以在 INSERT INTO 子句中列出每一个要插入值的列的名字。其数值的数据类型一定与对应的列的数据类型相匹配。

注意：

如果您在向某个表中插入中文字符时遇到了困难，不用着急。这很可能是在中文方式下输入引号或逗号造成的，您只要保证在英文方式下输入标点符号就应该没有问题了。

您也可以按表中列的默认顺序在 VALUES 子句中直接列出每一个要插入的数据值，而不用在 INSERT INTO 子句中列出每一个要插入值的列的名字，如例 12-7 所示。

例 12-7

```
SQL> INSERT INTO dept_dml
  2  VALUES            (77, '订货', '狼山市');
```

例 12-7 结果

```
已创建 1 行。
```

当向 dept_dml 表中插入该行数据之后，您可以使用例 12-8 的查询语句来检查例 12-7 的 DML 语句是否正确。

例 12-8

```
SQL> SELECT *
  2  FROM dept_dml;
```

例 12-8 结果

```
 DEPTNO DNAME          LOC
---------- -------------- ---------
       10 ACCOUNTING          NEW YORK
       20 RESEARCH            DALLAS
       30 SALES               CHICAGO
```

```
      40 OPERATIONS          BOSTON
      66 美容                煤球胡同
      77 订货                狼山市
已选择 6 行。
```

尽管使用例 12-7 的方法可以达到向一个表中插入一行数据的目的，但是在开发软件时应该列出所有要插入数据的列名，这样会使软件的易读性大为增加，也使软件更容易维护。

12.3　INSERT语句中的空值问题

通过前面几章的学习，您可能已经注意到了查询语句中的空值（NULL）一直是一个比较令人头痛的问题。如果处理不当可能会产生意想不到的结果，而且由于空值（NULL）产生的问题不属于语法错误，所以系统不会产生任何出错提示信息。在使用 INSERT 语句时也会遇到处理空值（NULL）的问题。下面我们通过两个例子来解释如何插入空值（NULL）。

您可以在 VALUES 子句中使用空串（''）向 dept_DML 表中输入一条含有空值（NULL）的记录，如例 12-9 的 DML 语句所示。

例 12-9

```
SQL> INSERT INTO dept_DML (deptno, dname, loc)
  2  VALUES              (88, '', '牛街');
```

例 12-9 结果

```
已创建 1 行。
```

使用例 12-9 的 DML 语句的目的是插入一条 dname 为空（NULL）的记录。它的商业背景可能是：公司决定成立一个新的部门，其部门号和地址都已确定，但是该部门的名字还没取好。

当使用例 12-9 的 DML 语句向 dept_dml 表中插入数据之后，可以使用例 12-10 的查询语句来检查所需的数据是否已经插入。

例 12-10

```
SQL> SELECT *
  2  FROM dept_DML;
```

例 12-10 结果

```
   DEPTNO DNAME                LOC
---------- -------------- --------
      10 ACCOUNTING          NEW YORK
      20 RESEARCH            DALLAS
      30 SALES               CHICAGO
      40 OPERATIONS          BOSTON
```

```
        66 美容                  煤球胡同
        77 订货                  狼山市
        88                       牛街
已选择 7 行。
```

从例 12-10 的显示结果中的最后一行可以看出，确实成功地插入了一条含有空值（NULL）的记录。

另外，也可以在 VALUES 子句中使用关键字 NULL 向 dept_DML 表中插入一条含有空值（NULL）的记录，如例 12-11 的 DML 语句。

例 12-11

```
SQL> INSERT INTO dept_DML (deptno, dname, loc)
  2  VALUES             (44, NULL, '安静大街');
```

例 12-11 结果

```
已创建 1 行。
```

之后，可以使用例 12-12 的查询语句来检查所需的数据是否已成功插入。

例 12-12

```
SQL> SELECT *
  2  FROM dept_DML;
```

例 12-12 结果

```
    DEPTNO DNAME                LOC
---------- -------------- ---------
        10 ACCOUNTING           NEW YORK
        20 RESEARCH             DALLAS
        30 SALES                CHICAGO
        40 OPERATIONS           BOSTON
        66 美容                  煤球胡同
        77 订货                  狼山市
        88                       牛街
        44                       安静大街
已选择 8 行。
```

从例 12-12 显示结果中的最后一行可以看出，您确实成功地插入了一条含有空值（NULL）的记录。

您还可以利用不列出要插入的列的方法向 dept_DML 表中插入一条含有空值（NULL）的记录，如例 12-13 的 DML 语句所示。

例 12-13

```
SQL> INSERT INTO dept_DML (deptno, dname)
  2  VALUES             (33, '公关');
```

例 12-13 结果

```
已创建 1 行。
```

之后，可以使用例 12-14 的查询语句来检查是否已成功地插入了这条最后一列为空值（NULL）的记录。

例 12-14

```
SQL> SELECT *
  2  FROM dept_DML;
```

例 12-14 结果

```
    DEPTNO DNAME                    LOC
---------- -------------- ---------
        10 ACCOUNTING               NEW YORK
        20 RESEARCH                 DALLAS
        30 SALES                    CHICAGO
        40 OPERATIONS               BOSTON
        66 美容                     煤球胡同
        77 订货                     狼山市
        88                          牛街
        44                          安静大街
        33 公关
已选择 9 行。
```

从例 12-14 的显示结果中的最后一行可以看出，您确实又一次成功地插入了一条含有空值（NULL）的记录。

现在，可以使用例 12-15 的 SQL语句来提交您刚插入的几行记录。

例 12-15

```
SQL> COMMIT;
```

例 12-15 结果

```
提交完成。
```

如果您不使用 SQL 的 COMMIT 语句提交这些刚插入的记录，这些记录可能会不被记录到 dept_DML 表中。

12.4　如何向表中插入特殊的值

可以在插入语句中使用特殊的值。例如，公司今天刚雇了一名新的保安，名字叫童铁蛋。现在经理让您将他的信息输入到员工表（emp_DML）中。很显然您没有必要在输入雇用日期（hiredate）时再重新输入今天的日期，您完全可以使用 Oracle 提供的系统日期函数（SYSDATE）来输入今天的日期，如例 12-16 所示。

例 12-16

```
SQL> INSERT INTO emp_DML (empno, ename, job, mgr,
  2                    hiredate, sal, comm, deptno)
  3  VALUES             (7800, '童铁蛋', '保安',7900, SYSDATE, 666, 77, 66);
```

例 12-16 结果

```
已创建 1 行。
```

为了可以在一行显示一条记录,您可以先使用例 12-17 的 SQL*Plus 命令来格式化输出。

例 12-17

```
SQL> col empno for 99999
SQL> col sal for 99999
SQL> col comm for 99999
SQL> col deptno for 99999
```

如果您的计算机显示器屏幕足够大,也可使用例 12-18 的 SQL*Plus 命令来格式化输出。

例 12-18

```
SQL> set line 100
```

之后，可以使用例 12-19 的查询语句来检查您是否已经成功地插入了这条含有特殊值（SYSDATE）的记录。

例 12-19

```
SQL> SELECT *
  2  FROM emp_dml;
```

例 12-19 结果

```
EMPNO ENAME      JOB       MGR  HIREDATE     SAL   COMM DEPTNO
------ ---------- --------- ---------- ---------- ------ ------ ---
  7369 SMITH      CLERK     7902 17-12 月-80   800          20
  7499 ALLEN      SALESMAN  7698 20-2 月 -81  1600   300    30
  7521 WARD       SALESMAN  7698 22-2 月 -81  1250   500    30
  7566 JONES      MANAGER   7839 02-4 月 -81  2975          20
  7654 MARTIN     SALESMAN  7698 28-9 月 -81  1250  1400    30
  7698 BLAKE      MANAGER   7839 01-5 月 -81  2850          30
  7782 CLARK      MANAGER   7839 09-6 月 -81  2450          10
  7788 SCOTT      ANALYST   7566 19-4 月 -87  3000          20
  7839 KING       PRESIDENT      17-11 月-81  5000          10
  7844 TURNER     SALESMAN  7698 08-9 月 -81  1500     0    30
  7876 ADAMS      CLERK     7788 23-5 月 -87  1100          20
  7900 JAMES      CLERK     7698 03-12 月-81   950          30
  7902 FORD       ANALYST   7566 03-12 月-81  3000          20
  7934 MILLER     CLERK     7782 23-1 月 -82  1300          10
  7800 童铁蛋     保安      7900 17-12 月-02   666    77    66
已选择 15 行。
```

从例 12-19 的显示结果中的最后一行可以看出，新员工童铁蛋的雇用日期（hiredate）确实是今天的日期（您的系统上的日期）。

12.5　如何利用子查询向表中插入数据

为了演示方便，首先使用例 12-20 的 SQL 语句创建一个名为 sales 的表。

例 12-20

```
SQL> CREATE TABLE sales (code, name, salary, commission)
   2  AS
   3  SELECT empno, ename, sal, comm
   4  FROM emp;
```

例 12-20 结果

```
表已创建。
```

之所以使用例 12-20 的 SQL 语句创建 sales 表，是因为使用这种方法创建 sales 表比较简单。其实，我们只需要 sales 表的结构。因此，可以使用例 12-21 的 SQL 语句删除 sales 表中的所有数据，而只保留该表的结构。

例 12-21

```
SQL> TRUNCATE TABLE sales;
```

例 12-21 结果

```
表已截掉。
```

现在可以使用例 12-22 的查询语句来检查是否已成功地创建了 sales 表结构，而该表中不包括任何数据。

例 12-22

```
SQL>  SELECT * FROM SALES;
```

例 12-22 结果

```
未选定行
```

现在可以利用子查询向 sales 表中插入数据，如例 12-23 的 INSERT 语句所示。

例 12-23

```
SQL> INSERT INTO sales (code, name, salary, commission)
   2          SELECT  empno, ename, sal, comm
   3          FROM emp
   4          WHERE job LIKE 'SALE%';
```

例 12-23 结果

```
已创建 4 行。
```

之后，可以使用例 12-24 的查询语句来检查是否已成功地插入了所有的记录。

例 12-24

```
SQL> SELECT * FROM SALES;
```

例 12-24 结果

```
      CODE NAME           SALARY COMMISSION
---------- ---------- ---------- ----------
      7499 ALLEN            1600        300
      7521 WARD             1250        500
      7654 MARTIN           1250       1400
      7844 TURNER           1500          0
```

注意：

- 在使用子查询向某个表中插入数据时不能使用 VALUES 关键字。
- INSERT 子句中的列数和数据类型必须与子查询中的列数和数据类型一致。

12.6 如何利用替代变量向表中插入数据和将INSERT语句存入脚本文件

如果只使用以上介绍的 INSERT 语句，将很难开发出商用软件。因为在每次插入数据时，我们都不得不重新改写 VALUES 子句中的值。向表中插入数据的另一个棘手的问题就是使用什么样的日期格式。如果选择的日期格式不合适，则当数据库中的字符集改变时，可能不得不重写所有选择使用日期格式的程序。您可以在 INSERT 语句中使用 SQL*Plus 的替代变量，这样 Oracle 就可以用交互的方式在 SQL 语句运行期间提示用户输入所需要列的值，如例 12-25 所示。

例 12-25

```
SQL> INSERT INTO emp_DML (empno, ename, job, mgr,
   2                      hiredate, sal, comm, deptno)
   3  VALUES              (&id, '&name', '&job', 7689,
   4                      TO_DATE('&hiredate', 'YYYY MM DD'),
   5                      666, 77, 66);
```

当运行例 12-25 的 SQL 语句时，Oracle 系统会给出输入的提示，此时可以输入所需的数值。按 Enter 键之后，Oracle 系统就会显示已创建 1 行。这说明 Oracle 系统已成功地插入了您所输入的记录。

例 12-25 系统提示和输入

输入 id 的值：1001

输入 name 的值：王老五

输入 job 的值：保洁员

输入 hiredate 的值：2002 07 08

注意：

阴影部分为系统提示，余下的部分为您输入的。

例 12-25 结果

```
已创建 1 行。
```

注意：

在例 12-25 的第 4 行中使用了 TO_DATE 函数，使用的模式为“年　月　日”，其中年为 4 位的阿拉伯数字，月为两位的阿拉伯数字，日也为两位的阿拉伯数字。之所以年、月、日都使用阿拉伯数字，是考虑到软件的可移植性，因为无论用户的数据库使用什么字符集都一定支持阿拉伯数字。之所以年份使用 4 位的阿拉伯数字而不用两位的阿拉伯数字，是为了避免“2000 年问题”。

现在可以使用例 12-26 的 SQL*Plus 命令将例 12-25 的 SQL 语句存入名为 d:\sql\insert.sql 的脚本文件。

例 12-26

```
SQL> save d:\sql\insert
```

例 12-26 结果

```
已创建文件 d:\sql\insert.sql
```

现在可以使用例 12-27 的查询语句来检查是否已成功地插入刚才输入的记录。

例 12-27

```
SQL> SELECT *
  2  FROM emp_dml;
```

例 12-27 结果

```
 EMPNO ENAME       JOB              MGR HIREDATE       SAL    COMM  DEPTNO
------ ----------- ---------- --------- ---------- ---- ------ --------
  7369 SMITH       CLERK           7902 17-12月-80    800              20
  7499 ALLEN       SALESMAN        7698 20-2月 -81   1600     300      30
  7521 WARD        SALESMAN        7698 22-2月 -81   1250     500      30
  7566 JONES       MANAGER         7839 02-4月 -81   2975              20
  7654 MARTIN      SALESMAN        7698 28-9月 -81   1250    1400      30
  7698 BLAKE       MANAGER         7839 01-5月 -81   2850              30
  7782 CLARK       MANAGER         7839 09-6月 -81   2450              10
  7788 SCOTT       ANALYST         7566 19-4月 -87   3000              20
  7839 KING        PRESIDENT            17-11月-81   5000              10
  7844 TURNER      SALESMAN        7698 08-9月 -81   1500       0      30
  7876 ADAMS       CLERK           7788 23-5月 -87   1100              20
  7900 JAMES       CLERK           7698 03-12月-81    950              30
```

```
7902 FORD       ANALYST      7566 03-12 月-81  3000          20
7934 MILLER     CLERK        7782 23-1 月 -82  1300          10
7800 童铁蛋     保安         7900 17-12 月-02   666    77    66
1001 王老五     保洁员       7689 08-7 月 -02   666    77    66
已选择 16 行。
```

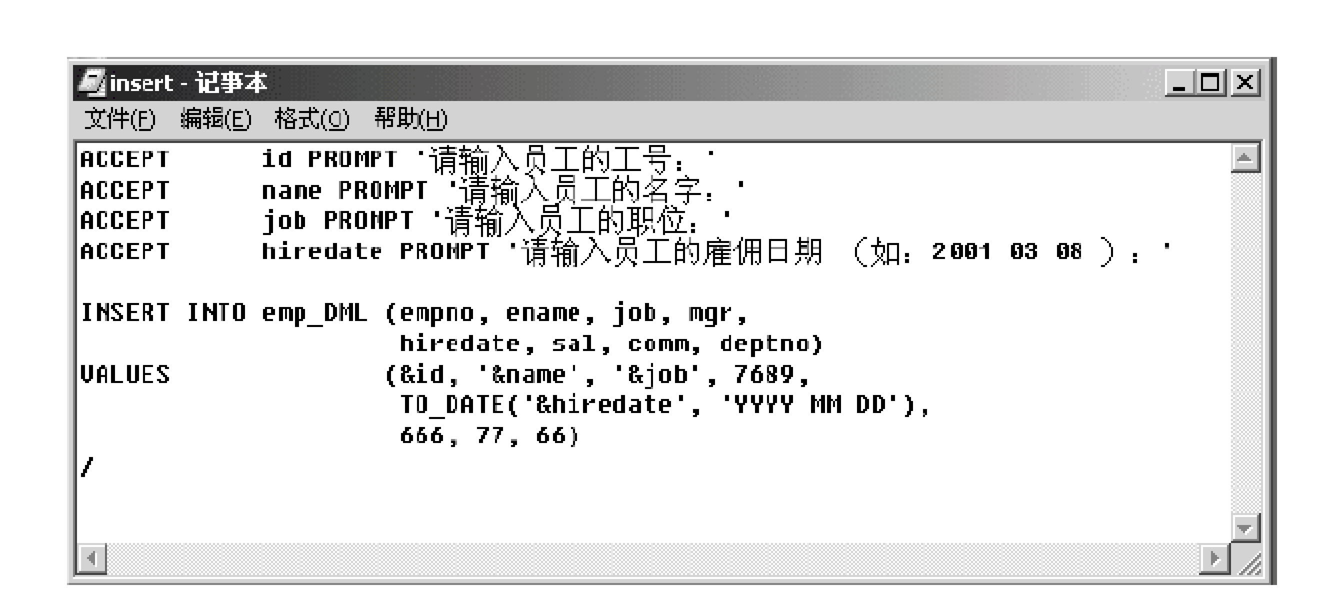
12.7 利用ACCEPT在INSERT语句中产生用户友好的系统提示

虽然使用例 12-25 的 SQL 语句可以开发商用软件，但是它产生的系统提示信息不是很清楚，普通用户不太容易理解这些提示信息。您可以使用 SQL*Plus 的 ACCEPT 命令重新“粉饰”一下例 12-25 的 SQL 语句以便产生用户友好的系统提示信息。为此，可以使用例 12-28 的 SQL*Plus 命令来编辑刚创建的脚本文件 d:\sql\insert.sql。

例 12-28

```
SQL> edit d:\sql\insert.sql
```

例 12-28 结果




```
ACCEPT      id PROMPT '请输入员工的工号：'
ACCEPT      name PROMPT '请输入员工的名字：'
ACCEPT      job PROMPT '请输入员工的职位：'
ACCEPT      hiredate PROMPT '请输入员工的雇佣日期 （如：2001 03 08 ）：'

INSERT INTO emp_DML (empno, ename, job, mgr,
                     hiredate, sal, comm, deptno)
VALUES              (&id, '&name', '&job', 7689,
                     TO_DATE('&hiredate', 'YYYY MM DD'),
                     666, 77, 66)
/
```

当编辑器打开 d:\sql\insert.sql 之后，可在 INSERT 语句之前插入例 12-28 结果所示的 4 行 ACCEPT 命令，之后存盘退出。现在可以使用例 12-29 所示的 SQL*Plus 命令来运行刚修改过的脚本文件。

例 12-29

```
SQL> @d:\sql\insert.sql
```

例 12-29 系统提示和输入

请输入员工的工号：1002
请输入员工的名字：吴秀刚
请输入员工的职位：保安
请输入员工的雇佣日期（如 2001 03 08）：2002 12 09

注意：

阴影部分为系统提示，余下部分为您输入的内容。

例 12-29 结果

```
已创建 1 行。
```

显然，例 12-29 的系统提示信息比例 12-25 的系统提示信息更清楚，也更加友好。现在您可以使用例 12-30 的查询语句来检查是否已成功地插入了刚才输入的记录。

例 12-30

```
SQL> SELECT *
2  FROM emp_dml;
```

例 12-30 结果

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12 月-80	800		20
7499	ALLEN	SALESMAN	7698	20-2 月 -81	1600	300	30
7521	WARD	SALESMAN	7698	22-2 月 -81	1250	500	30
7566	JONES	MANAGER	7839	02-4 月 -81	2975		20
7654	MARTIN	SALESMAN	7698	28-9 月 -81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-5 月 -81	2850		30
7782	CLARK	MANAGER	7839	09-6 月 -81	2450		10
7788	SCOTT	ANALYST	7566	19-4 月 -87	3000		20
7839	KING	PRESIDENT		17-11 月-81	5000		10
7844	TURNER	SALESMAN	7698	08-9 月 -81	1500	0	30
7876	ADAMS	CLERK	7788	23-5 月 -87	1100		20
7900	JAMES	CLERK	7698	03-12 月-81	950		30
7902	FORD	ANALYST	7566	03-12 月-81	3000		20
7934	MILLER	CLERK	7782	23-1 月 -82	1300		10
7800	童铁蛋	保安	7900	17-12 月-02	666	77	66
1001	王老五	保洁员	7689	08-7 月 -02	666	77	66
1002	吴秀刚	保安	7689	09-12 月-02	666	77	66

已选择 17 行。

12.8　UPDATE语句

还记得在第 2 章开始的例子吗？现在我们假设老板是一位非常仁慈的人，虽然公司因受互联网泡沫破裂和 9.11 恐怖袭击的双重打击，已经连续几个季度亏损，老板还是不忍心解雇任何员工，但是公司已不堪重负，最后老板决定公司与员工“同舟共济”，所有的员工都减薪百分之十。老板让您来修改数据库中员工的工资，您可以使用例 12-31 的 UPDATE 语句来完成老板的重托。

例 12-31

```
SQL> UPDATE emp_dml
    2  SET sal = sal * 0.9;
```

例 12-31 结果

```
已更新 17 行。
```

现在您可以使用例 12-32 的查询语句来检查是否已成功地使每个员工的工资（sal）减少了百分之十。

例 12-32

```
SQL> SELECT ename, job, sal
  2  FROM emp_dml;
```

例 12-32 结果

```
ENAME                JOB                SAL
---------- --------- ----------
SMITH                CLERK              720
ALLEN                SALESMAN          1440
WARD                 SALESMAN          1125
JONES                MANAGER         2677.5
MARTIN               SALESMAN          1125
BLAKE                MANAGER           2565
CLARK                MANAGER           2205
SCOTT                ANALYST           2700
KING                 PRESIDENT         4500
TURNER               SALESMAN          1350
ADAMS                CLERK              990
JAMES                CLERK              855
FORD                 ANALYST           2700
MILLER               CLERK             1170
童铁蛋               保安              599.4
王老五               保洁员            599.4
吴秀刚               保安              599.4
已选择 17 行。
```

从例 12-32 的显示结果可以看出，emp_dml 表中的每一名员工的工资确实都减少了百分之十。

UPDATE 语句的格式如下：

UPDATE　表名

SET　　　列名=值 [, 列名=值, …]

[WHERE　条件];

其中各参数的含义如下。

- 表名：要修改数据的表的名字。
- 列名：表中要修改数据列的名字。
- 数值：对应列的具体值。
- 条件：由列名、表达式、常量、子查询和比较运算符组成，用来决定所要修改的数据行。

与 INSERT 语句不同，UPDATE 语句可以一次修改多行记录。因此，如果您要修改一条记录的话，在写 UPDATE 语句时就要保证一定只有一条记录能满足 WHERE 子句中的条件。为了解释这一点，我们在 emp_dml 中插入了另一名也叫童铁蛋的员工。不过他的职位、工资与前一个童铁蛋大不相同。下面使用例 12-33 的查询语句来查看他们的信息。

例 12-33

```
SQL> select * from emp_dml;
```

例 12-33 结果

```
     EMPNO ENAME      JOB              MGR HIREDATE          SAL       COMM
---------- ---------- --------- ---------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-12 月-80       800
      7499 ALLEN      SALESMAN        7698 20-2 月 -81      1600        300
      7521 WARD       SALESMAN        7698 22-2 月 -81      1250        500
      7566 JONES      MANAGER         7839 02-4 月 -81      2975
      7654 MARTIN     SALESMAN        7698 28-9 月 -81      1250       1400
      7698 BLAKE      MANAGER         7839 01-5 月 -81      2850
      7782 CLARK      MANAGER         7839 09-6 月 -81      2450
      7788 SCOTT      ANALYST         7566 19-4 月 -87      3000
      7839 KING       PRESIDENT            17-11 月-81      5000
      7844 TURNER     SALESMAN        7698 08-9 月 -81      1500          0
      7876 ADAMS      CLERK           7788 23-5 月 -87      1100
      7900 JAMES      CLERK           7698 03-12 月-81       950
      7902 FORD       ANALYST         7566 03-12 月-81      3000
      7934 MILLER     CLERK           7782 23-1 月 -82      1300
      7800 童铁蛋     保安            7900 17-12 月-02       666         77
      7810 童铁蛋     CTO             7900 17-12 月-02      4444
      1001 王老五     保洁员          7689 08-7 月 -02       666         77
      1002 吴秀刚     保安            7689 09-12 月-02       666         77
已选择 18 行。
```

从例 12-33 的显示结果可以看出，在 emp_dml 表中有两个叫童铁蛋的员工，一个为保安，其工资只为 666 元，另一个为 CTO（技术总监），其工资为 4444 元。

假设公司的保安童铁蛋为保护公司的财产在与歹徒们的英勇搏斗中光荣负伤，从而阻止了一次重大的偷窃事件。公司老板为了表彰他的勇敢行为，也是为了让公司的其他员工争先效法为公司卖命，老板决定将他的工资从 666 元即刻提升为 1000 元。老板再一次要求您修改数据库中相关的信息，您可以试着用例 12-34 的 DML 语句来完成他交给您的使命。

例 12-34

```
SQL> UPDATE emp_dml
    2  SET sal = 1000
    3  WHERE ename = '童铁蛋';
```

例 12-34 结果

```
已更新 2 行。
```

当看到例 12-34 的显示结果时，您可能会感到不安。因为 Oracle 系统显示已更新 2 行，而老板只决定将见义勇为的保安童铁蛋一人的工资增加到 1000 元。现在您可以使用例 12-35 的查询语句来看看究竟发生了什么。

例 12-35

```
SQL> SELECT *
    2  FROM emp_dml;
```

例 12-35 结果

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7369	SMITH	CLERK	7902	17-12 月-80	800	
7499	ALLEN	SALESMAN	7698	20-2 月 -81	1600	300
7521	WARD	SALESMAN	7698	22-2 月 -81	1250	500
7566	JONES	MANAGER	7839	02-4 月 -81	2975	
7654	MARTIN	SALESMAN	7698	28-9 月 -81	1250	1400
7698	BLAKE	MANAGER	7839	01-5 月 -81	2850	
7782	CLARK	MANAGER	7839	09-6 月 -81	2450	
7788	SCOTT	ANALYST	7566	19-4 月 -87	3000	
7839	KING	PRESIDENT		17-11 月-81	5000	
7844	TURNER	SALESMAN	7698	08-9 月 -81	1500	0
7876	ADAMS	CLERK	7788	23-5 月 -87	1100	
7900	JAMES	CLERK	7698	03-12 月-81	950	
7902	FORD	ANALYST	7566	03-12 月-81	3000	
7934	MILLER	CLERK	7782	23-1 月 -82	1300	
7800	童铁蛋	保安	7900	17-12 月-02	1000	77
7810	童铁蛋	CTO	7900	17-12 月-02	1000	
1001	王老五	保洁员	7689	08-7 月 -02	666	77
1002	吴秀刚	保安	7689	09-12 月-02	666	77

已选择 18 行。

从例 12-35 的显示结果可以清楚地看出，在 emp_dml 表中除了保安童铁蛋的工资被改为 1000 元之外，技术总监（CTO）的工资也被改为 1000 元，这显然是一个天大的笑话。之所以发生这种事情，是因为在 emp_dml 表中员工的名字（ename）可以重名造成的。为了避免这类事情的发生，您应该保证 WHERE 子句中的条件能唯一地标识表中的一行，最

稳妥的办法是使用主键（Primary Key）。

☠ **注意：**

在使用 UPDATE 语句修改表中的记录时，如果 WHERE 子句中的条件写错了，就会造成该修改的记录没改，而不该修改的记录都改了。如果忘写了 WHERE 子句，就会造成表中所有的记录都被修改。所以在使用 UPDATE 语句时要非常小心，一定要确保 WHERE 子句中的条件是正确的。

12.9　基于另一个表来修改记录

假设公司的人事经理是刚刚走马上任的，他为了节约公司开销，也为了取乐老板开始狠狠地压低新员工的工资。还记得公司刚雇的几名新员工吗？他们的起薪都只有 666 元。老板发现新员工的工资都很低自然高兴。高兴之余也有几分担忧，因为老板担心这些员工的起薪低于公司所允许的最低工资。如果真的是这样的话，让工会抓到了把柄可就吃不了兜着走了。老板赶紧让您核查一下。您可以使用例 12-36 的查询语句从工资级别（salgrade）表中得到相关信息。

例 12-36

```
SQL> SELECT *
  2  FROM salgrade;
```

例 12-36 结果

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

salgrade 表的第 1 列（GRADE）为职位的等级，第 2 列（LOSAL）为该级别的最低工资，第 3 列（HISAL）为该级别的最高工资。

从例 12-36 的显示结果可以清楚地看出，果然不出老板所料，这些新员工的工资已经低于公司所允许的最低工资 700 元。当您把这一结果告诉老板时，他老人家摇着头叫您以最快的速度将公司中所有低于公司所允许的最低工资的员工的工资改为公司所允许的最低工资（看来这位人事经理的马屁没拍到点上），以免让工会告到法庭上。您可以使用例 12-37 的 UPDATE 语句来修改表中相关的记录。

例 12-37

```
SQL> UPDATE emp_dml
  2  SET sal = (SELECT losal
```

```
          FROM   salgrade
          WHERE  grade = 1)
WHERE sal < (SELECT losal
             FROM   salgrade
             WHERE  grade = 1);
```

例 12-37 结果

```
已更新 3 行。
```

在例 12-37 的 UPDATE 语句中，除了在 WHERE 子句中使用了子查询，您还在 SET 子句中使用了子查询并把该子查询语句的返回值赋给了 sal。

可以在 UPDATE 语句中利用基于另一个表的子查询来修改表中的记录。

现在可以使用例 12-38 的查询语句来检查一下例 12-37 所做的修改是否正确。

例 12-38

```
SQL> SELECT *
  FROM emp_dml;
```

例 12-38 结果

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7369	SMITH	CLERK	7902	17-12 月-80	800	
7499	ALLEN	SALESMAN	7698	20-2 月 -81	1600	300
7521	WARD	SALESMAN	7698	22-2 月 -81	1250	500
7566	JONES	MANAGER	7839	02-4 月 -81	2975	
7654	MARTIN	SALESMAN	7698	28-9 月 -81	1250	1400
7698	BLAKE	MANAGER	7839	01-5 月 -81	2850	
7782	CLARK	MANAGER	7839	09-6 月 -81	2450	
7788	SCOTT	ANALYST	7566	19-4 月 -87	3000	
7839	KING	PRESIDENT		17-11 月-81	5000	
7844	TURNER	SALESMAN	7698	08-9 月 -81	1500	0
7876	ADAMS	CLERK	7788	23-5 月 -87	1100	
7900	JAMES	CLERK	7698	03-12 月-81	950	
7902	FORD	ANALYST	7566	03-12 月-81	3000	
7934	MILLER	CLERK	7782	23-1 月 -82	1300	
7800	童铁蛋	保安	7900	17-12 月-02	700	77
7810	童铁蛋	CTO	7900	17-12 月-02	4444	
1001	王老五	保洁员	7689	08-7 月 -02	700	77
1002	吴秀刚	保安	7689	09-12 月-02	700	77

已选择 18 行。

从例 12-38 的结果可以看出，emp_dml 表中所有职位为保安和保洁员的员工的工资都增加到了 700 元。

12.10　利用多列子查询来修改记录

假设公司又要重组，将取消保安这一职位并将具有此头衔的员工的职位改为文员（CLERK），这样那些保安没事干时就可干一些简单的文员工作。但是文员的最低工资为 800 元。工会对公司的一个基本要求就是员工要同工同酬，公司的高级管理层当然不想惹怒工会。于是又再一次要求您修改数据库，将所有保安的职位改为文员，并且将他们的工资提高到文员的最低工资。

您知道一个叫 SMITH 的文员工资最低，他的工号（EMPNO）为 7369。于是，您使用例 12-39 的 UPDATE 语句来修改表中所有的职位为保安的员工的相关信息。

例 12-39

```
SQL> UPDATE emp_dml
  2  SET    (job, sal) = (SELECT job, sal
  3                       FROM emp_dml
  4                       WHERE empno = 7369)
  5  WHERE job = '保安';
```

例 12-39 结果

```
已更新 2 行。
```

可以使用 UPDATE 语句来同时修改表中的多列。但是在使用这种方法时，在 SET 子句中等号右边的必须是子查询语句。

现在您可以使用例 12-40 的查询语句来检查例 12-39 所做的修改是否已经正确地完成。

例 12-40

```
SQL> SELECT *
  2  FROM emp_dml;
```

例 12-40 结果

```
     EMPNO ENAME      JOB              MGR HIREDATE          SAL       COMM
---------- ---------- ---------- ---------- ---------- ---------- --------
      7369 SMITH      CLERK           7902 17-12 月-80        800
      7499 ALLEN      SALESMAN        7698 20-2 月 -81       1600        300
      7521 WARD       SALESMAN        7698 22-2 月 -81       1250        500
      7566 JONES      MANAGER         7839 02-4 月 -81       2975
      7654 MARTIN     SALESMAN        7698 28-9 月 -81       1250       1400
      7698 BLAKE      MANAGER         7839 01-5 月 -81       2850
      7782 CLARK      MANAGER         7839 09-6 月 -81       2450
      7788 SCOTT      ANALYST         7566 19-4 月 -87       3000
      7839 KING       PRESIDENT            17-11 月-81       5000
      7844 TURNER     SALESMAN        7698 08-9 月 -81       1500          0
      7876 ADAMS      CLERK           7788 23-5 月 -87       1100
```

7900	JAMES	CLERK	7698	03-12 月-81	950	
7902	FORD	ANALYST	7566	03-12 月-81	3000	
7934	MILLER	CLERK	7782	23-1 月 -82	1300	
7800	童铁蛋	CLERK	7900	17-12 月-02	800	77
7810	童铁蛋	CTO	7900	17-12 月-02	4444	
1001	王老五	保洁员	7689	08-7 月 -02	700	77
1002	吴秀刚	CLERK	7689	09-12 月-02	800	77

已选择 18 行。

从例 12-40 的结果可以看出，emp_dml 表中所有职位为保安的员工的职位都已修改为文员（CLERK），并且工资都增加到了 800 元。

12.11 DELETE语句

虽然您的老板是一位大慈大悲的“活菩萨”，但公司经过员工集体减薪百分之十和结构重组后仍然持续亏损，老板要是不采取非常手段的话公司可能只有倒闭。老板出于无奈，为了大多数员工和股东们的利益不得不开“杀戒”，即解雇一批员工。

为此，公司的董事们开了一次决定公司生死存亡的重要会议。在这次充满了火药味的会议上，老板想当“和事佬”也不行了，董事们最终一致认为：“那些拿高薪的经理们都是些笨蛋，应该最先赶走。那些拿高薪的推销员也是废物，也得滚蛋。”

您又一次成了这个重大决议的执行者。根据董事会的决议，老板通知您将工资高于 2500 元的经理和工资高于 1300 元的推销员的记录从数据库中删除掉（即炒鱿鱼）。为了将来验证方便，您使用了例 12-41 的查询语句来查看一下在 emp_dml 表中所有的经理和推销员的记录，并按工资由低到高排序（也是想最后看看您的那些老领导和老同事们，因为下一波大裁员还不知道轮到谁呢）。

例 12-41

```
SQL> SELECT *
  2  FROM emp_dml
  3  WHERE (job = 'MANAGER') OR (JOB = 'SALESMAN')
  4  ORDER BY sal;
```

例 12-41 结果

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
---------	---------	--------	----------	----------	----------	--------
7521	WARD	SALESMAN	7698	22-2 月 -81	1250	500
7654	MARTIN	SALESMAN	7698	28-9 月 -81	1250	1400
7844	TURNER	SALESMAN	7698	08-9 月 -81	1500	0
7499	ALLEN	SALESMAN	7698	20-2 月 -81	1600	300
7782	CLARK	MANAGER	7839	09-6 月 -81	2450	
7698	BLAKE	MANAGER	7839	01-5 月 -81	2850	

```
    7566 JONES     MANAGER        7839 02-4 月 -81       2975
已选择 7 行。
```

当您最后一次仔细地浏览了例 12-41 结果中那些熟悉和亲切的信息之后，忧伤而不情愿地执行了例 12-42 的 DELETE 语句。

例 12-42

```
SQL> DELETE FROM emp_dml
  2  WHERE (job = 'MANAGER' AND sal > 2500)
  3  OR    (JOB = 'SALESMAN' AND sal > 1300);
```

例 12-42 结果

```
已删除 4 行。
```

尽管您非常不情愿，但还是应该使用例 12-43 的查询语句来检查一下例 12-42 所做的删除是否已正确地执行。

例 12-43

```
SQL>  SELECT *
  2   FROM emp_dml
  3   WHERE (job = 'MANAGER') OR (JOB = 'SALESMAN')
  4   ORDER BY sal;
```

例 12-43 结果

```
     EMPNO ENAME      JOB              MGR HIREDATE          SAL       COMM
---------- ---------- ---------- ---------- ---------- ---------- --------
      7521 WARD       SALESMAN        7698 22-2 月 -81       1250        500
      7654 MARTIN     SALESMAN        7698 28-9 月 -81       1250       1400
      7782 CLARK      MANAGER         7839 09-6 月 -81       2450
```

当您仔细地浏览例 12-43 显示的结果时，许多熟悉的同志和战友们已经不见了，您感到很内疚。正是由于您所执行的这条 DML 语句使您的同事和朋友的买房梦成了泡影；使您的同事和朋友不得不变卖象征他/她们身份和地位的汽车。最让您感到毛骨悚然的是，由于您所发的这条 DML 语句使您的一位好朋友的红颜知己离他而去。原来生活是那样的冷酷，又是那样的现实。

DELETE 语句用来删除一个表中存在的行。

该语句的格式如下：

DELETE　[FROM]　表名

[WHERE　条件];

其中各参数的含义如下。

- 表名：要从中删除数据的表名。
- 条件：由列名、表达式、常量、子查询和比较运算符组成，用来决定所要修改的数据行。

尽管从 SQL 语句的易读性来讲应该使用 FROM 关键字，但许多有经验的 Oracle 专业

人士都不使用这一关键字。这可能是为了减少输入量。像 UNIX 和 C 的专业人士一样，Oracle 专业人士也喜欢使用缩写形式。上帝也许是公平的，因为软件人员的逻辑思维都很好但打字的速度则较慢（既有所长，就一定有所短）。

12.12　在使用DELETE时可能出现的问题

与 UPDATE 语句相似，DELETE 语句可以删除多条记录。如果您在执行例 12-42 的 DELETE 语句时将 WHERE 子句中的一个条件写反了，写成了例 12-44 的 DML 语句。

例 12-44

```
SQL> DELETE FROM emp_dml
  2  WHERE (job = 'MANAGER' AND sal < 2500)
  3   OR   (JOB = 'SALESMAN' AND sal > 1300);
```

例 12-44 结果

```
已删除 3 行。
```

现在您应该使用例 12-45 的查询语句来检查是否执行了正确的删除操作。

例 12-45

```
SQL> SELECT *
  2  FROM emp_dml
  3  WHERE (job = 'MANAGER') OR (JOB = 'SALESMAN')
  4  ORDER BY sal;
```

例 12-45 结果

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7521	WARD	SALESMAN	7698	22-2 月 -81	1250	500
7654	MARTIN	SALESMAN	7698	28-9 月 -81	1250	1400
7698	BLAKE	MANAGER	7839	01-5 月 -81	2850	
7566	JONES	MANAGER	7839	02-4 月 -81	2975	

当您看到例 12-45 的显示结果时，可能大吃一惊。因为该炒鱿鱼的经理们都还在数据库中，而应该留住的经理们又都被炒鱿鱼了。如果您忘了写 WHERE 子句又会发生什么呢？例如，您不小心执行了例 12-46 的 DML 语句。

例 12-46

```
SQL> DELETE emp_dml;
```

例 12-46 结果

```
已删除 18 行。
```

当看到例 12-46 的显示结果时，您感到事情不妙，可以使用例 12-47 的查询语句来查看到底删除了多少行记录。

例 12-47

```
SQL> SELECT *
  2  FROM emp_dml;
```

例 12-47 结果

```
未选定行
```

例 12-47 的显示结果表明，您已把 emp_dml 中的所有记录都删除了。

从例 12-44~例 12-47 的演示您可能已经得出了结论：在使用 DELETE 语句时一定要确保 WHERE 子句中的条件正确。因此，在执行 DELETE 语句之前要反复检查 WHERE 子句中的条件是否正确。如果您管理的是一个司法系统数据库，在使用 DELETE 语句时就应更加小心，因为这时您握有“生杀大权”。

12.13　基于另一个表来删除行

您可以在 WHERE 子句中利用子查询删除基于另外的表的行。假设公司又进行了进一步的重组。考虑到美容这行当越来越难做，满大街都有美容美发，竞争实在太激烈了。因此，公司决定即刻退出这个市场，同时，要把美容部门的全部职工优化组合掉。您再一次受命于危难之时，去做那让您想起来就胆颤心寒的工作。您使用了例 12-48 的 DML 语句。

例 12-48

```
SQL> DELETE FROM emp_dml
  2  WHERE        deptno =
  3                    (SELECT deptno
  4                     FROM dept_dml
  5                     WHERE dname = UPPER('美容'));
```

例 12-48 结果

```
已删除 4 行。
```

现在您应该使用例 12-49 的查询语句来检查是否正确地删除了要删除的记录。

例 12-49

```
SQL> SELECT ename, job, sal, comm, deptno
  2  FROM emp_dml;
```

例 12-49 结果

```
ENAME      JOB              SAL       COMM     DEPTNO
---------- --------- ---------- ---------- ------
SMITH      CLERK            800                  20
ALLEN      SALESMAN        1600        300       30
WARD       SALESMAN        1250        500       30
JONES      MANAGER         2975                  20
```

```
MARTIN      SALESMAN     1250        1400          30
BLAKE       MANAGER      2850                      30
CLARK       MANAGER      2450                      10
SCOTT       ANALYST      3000                      20
KING        PRESIDENT    5000                      10
TURNER      SALESMAN     1500           0          30
ADAMS       CLERK        1100                      20
JAMES       CLERK         950                      30
FORD        ANALYST      3000                      20
MILLER      CLERK        1300                      10
已选择 14 行。
```

从例 12-49 的显示结果可以看出，所有 66 号部门的员工记录都不见了。

如果您比较细心的话，可能早已发现了一些您修改过和删除的记录依然在数据库中保持不变。这是因为 Oracle 的事务（Transactions）处理控制机制在发挥作用。

12.14 引入事务处理的原因

假设某人在一银行开了两个账户，一个账户为活期储蓄，另一个为支票。一般活期储蓄的利息比支票账户的高。为了多赚点利息他几乎把全部的钱都存在了活期储蓄账户中。假设此人在活期储蓄账户中存了 8888 元，而在支票账户中只存了 30 元。现在他报名参加一个 Oracle 培训课程要交 6800 元的培训费，他想用支票来交学费，因此，他不得不先从活期储蓄账户转 6800 元到支票账户中。如果把他在银行的自动提款机（ATM）上的操作转换成 Oracle 数据库的命令，则操作如下：

（1）查询活期储蓄账户中的钱数。

（2）查询支票账户中的钱数。

（3）从活期储蓄账户中减掉 6800 元。

（4）在支票账户中加上 6800 元。

如果银行的数据库正好在第 3 步之后崩溃了，该客户就可能损失 6800 元。如果有这样的银行数据库系统的话，相信没人敢向该银行里存钱。因此，Oracle 数据库管理系统必须能正确地处理这种情况。

如果 Oracle 服务器要保证数据库中的数据是一致的，它就必须保证以上的第 3 和第 4 步操作要么都完成，要么都放弃，即这两个操作在逻辑上是一个不可分割的整体。在 Oracle 数据库中称之为事务（Transactions）。

12.15 什么是Oracle数据库的事务

为了有效地控制事务（Transactions），Oracle 引入了两个显式的事务控制命令（语句），

一个是 COMMIT，另一个是 ROLLBACK。

Oracle 数据库的事务可由：

- 一个或多个 DML 语句组成。
- 一个 DDL 语句组成。
- 一个 DCL 语句组成。

那么如何标识一个事务（Transactions）呢？因为 Oracle 公司在开始时是瞄准联机事务处理（OLTP）的，所以它对事务处理提供了强有力的支持。Oracle 可以自动地标识一个事务。

Oracle 的一个事务是以第 1 个可执行的 SQL 语句开始，当下列事件之一发生时结束。

- 用户执行了 COMMIT 语句（提交）。
- 用户执行了 ROLLBACK 语句（回滚）。
- 用户执行了 DDL 语句（自动提交）。
- 用户执行了 DCL 语句（自动提交）。
- 用户正常退出 SQL*Plus（自动提交）。
- 用户非正常退出 SQL*Plus（自动回滚）。
- 系统崩溃，包括硬件或软件故障（自动回滚）。

作为一位 Oracle 的专业人员，您应该尽可能地使用 COMMIT 和 ROLLBACK 语句来显式地控制事务的提交和回滚。因为使用后 5 条隐式事务控制特性有时可能会产生意想不到的结果。现在我们可以使用这两个语句把 12.14 节中在银行的自动提款机（ATM）上那些逻辑上相关的操作集成为一个事务。这些逻辑上相关的操作要么全部完成（提交），要么全部放弃（回滚）。

12.16 利用COMMIT和ROLLBACK语句进行事务控制

在 Oracle 数据库系统上执行事务处理的用户可以看到正在修改的数据，而其他的用户只能看到修改之前的数据。一旦该事务被提交，所有的用户都能看到修改之后的数据。下面用几个例子来演示如何利用 COMMIT 和 ROLLBACK 语句来控制事务处理。

首先用 SCOTT 用户登录数据库。假设老板对数字很在意，他让您把第 66 号部门改成 88 号，您可以使用例 12-50 的 DML 语句来完成他交给您的任务。

例 12-50

```
SQL> UPDATE emp_dml
  2  SET deptno = 88
  3  WHERE deptno = 66;
```

例 12-50 结果

```
已更新 4 行。
```

作为一位资深的 Oracle 数据库管理员，您做事非常谨慎，虽然 Oracle 数据库系统已经提示："已更新 4 行。"但您还是想用例 12-51 的查询语句来验证所做的修改是否准确无误。

例 12-51

```
SQL> SELECT empno, ename, deptno
  2  FROM emp_dml
  3  WHERE deptno > 30;
```

例 12-51 结果

```
     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7800 童铁蛋             88
      7810 童铁蛋             88
      1001 王老五             88
      1002 吴秀刚             88
```

从例 12-51 的显示结果可以看出，您已经按老板的要求成功地修改了部门号，您立即打电话通知老板。接到您的电话，他使用另一个用户名登录数据库（如 SYSTEM/MANAGER）。他使用了与您在例 12-51 中几乎相同的查询语句例 12-52 来查看您刚做过的修改。

例 12-52

```
SQL> SELECT empno, ename, deptno
    2  FROM scott.emp_dml
    3  WHERE deptno > 30;
```

例 12-52 结果

```
     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7800 童铁蛋             66
      7810 童铁蛋             66
      1001 王老五             66
      1002 吴秀刚             66
```

可是令他吃惊的是，例 12-52 的显示结果表明第 66 号部门没有任何变化，于是他赶紧打电话询问您其中的原因。

这时您意识到事务还没有提交，于是您立刻使用例 12-53 的 COMMIT 命令来提交您的事务。

例 12-53

```
SQL> COMMIT;
```

例 12-53 结果

```
提交完成。
```

您再一次打电话通知老板修改已经成功。接到您的电话，他重新执行了例 12-52 的查询语句（例 12-54）来查看您刚做过的修改。

例 12-54

```
SQL> SELECT empno, ename, deptno
  2  FROM scott.emp_dml
  3  WHERE deptno > 30;
```

例 12-54 结果

```
     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7800 童铁蛋             88
      7810 童铁蛋             88
      1001 王老五             88
      1002 吴秀刚             88
```

这回您没有让老板失望，他终于看到了他所喜爱的部门号 88。

12.17　利用DDL和DCL语句进行事务控制

还记得在 12.9 节中的例子吗？您的老板为了避免惹麻烦，他让您以最快的速度将公司中所有低于公司所允许的最低工资的员工的工资改为公司所允许的最低工资。您也使用了 UPDATE 语句来修改表中相关的记录，但您忘了使用 COMMIT 语句提交您所做的修改。

也许是刚过完春节，您的老板想图个吉利，因此，他让您把那些低于公司所允许的最低工资的员工的工资改为 800 元而不是原来的 700 元。因为您知道所有的低于公司所允许的最低工资的员工都是新雇的且都在第 88 号部门，为了简单起见，所以您使用了例 12-55 的 DML 语句来修改这些员工的工资。

例 12-55

```
SQL> UPDATE emp_dml
  2  SET sal = 800
  3  WHERE deptno = 88;
```

例 12-55 结果

```
已更新 4 行。
```

您做事非常谨慎，虽然 Oracle 数据库系统已经提示“已更新 4 行”，作为一位“久经沙场”的 Oracle 专家，您还是使用了例 12-56 的查询语句来验证所做的修改是否准确无误。

例 12-56

```
SQL> SELECT ename, job, sal
  2  FROM emp_dml
  3  WHERE deptno = 88;
```

例 12-56 结果

```
ENAME      JOB         SAL
---------- --------- -------
童铁蛋     MANAGER       800
童铁蛋     CTO           800
王老五     保洁员        800
吴秀刚     保安          800
```

有过上一次的教训，您变得更加谨慎。您并没有立即通知老板，而是使用另一个用户名登录数据库（如 SYSTEM/MANAGER），之后，使用例 12-57 的查询语句来验证您所做的修改是否已写到数据库中。

例 12-57

```
SQL> SELECT ename, job, sal
    2  FROM scott.emp_dml
    3  WHERE deptno = 88;
```

例 12-57 结果

```
ENAME      JOB         SAL
---------- --------- -------
童铁蛋     MANAGER      666
童铁蛋     CTO          4444
王老五     保洁员       666
吴秀刚     保安         666
```

例 12-57 显示的结果清楚地表明您所做的修改没有写到数据库中，这时，您想起还没有执行 COMMIT 命令。而此时您正需要使用例 12-58 的 DDL 语句来创建一个名为 dept_ddl 的临时表（在 SCOTT 用户中），因此，也就没有必要再使用 COMMIT 命令来提交您所做的修改了。

例 12-58

```
SQL> CREATE TABLE dept_ddl
   2  AS
   3  SELECT *
   4  FROM dept;
```

例 12-58 结果

```
表已创建。
```

为了保险起见，您在 SYSTEM 用户名下又重新使用了刚使用过的查询语句（例 12-59）来验证这次所做的修改是否真的已写到数据库中。

例 12-59

```
SQL> SELECT ename, job, sal
    2  FROM scott.emp_dml
    3  WHERE deptno = 88;
```

例 12-59 结果

```
ENAME      JOB       SAL
---------- --------- ----------
童铁蛋     MANAGER        800
童铁蛋     CTO            800
王老五     保洁员         800
吴秀刚     保安           800
```

看过例 12-59 的显示结果您可以放心地通知老板了（如果您细心的话，可能已经注意到了职位（job）为 CTO 的童铁蛋的工资被错误地减少了 3000 多元。您能给出正确的 UPDATE 语句吗？）。

在这一节中我们并没有给出用 DCL 语句结束事务的例子，其原因有两条：

（1）到目前为止我们还没有介绍什么是 DCL 语句。

（2）用 DCL 语句结束事务的操作与用 DDL 语句结束事务的操作几乎完全相同。

基于以上的理由，读者学会了 DCL 语句后就能很容易地构造出利用 DCL 语句结束事务的例子。

12.18　非正常退出和正常退出SQL*Plus对事务控制的影响

如果您读过其他的 Oracle SQL 的书，可能会发现书中只提到用户退出 SQL*Plus 会结束事务。也有的书中写到用户正常退出 SQL*Plus，系统会自动提交事务。那么什么叫正常退出 SQL*Plus，什么又叫非正常退出 SQL*Plus 呢？我们用下面的例子来分别演示这两种情形。

假设您用例 12-60 的 DML 语句将王老五的职位（job）改为经理（MANAGER），虽然只是一个光杆司令。

例 12-60

```
SQL> UPDATE emp_dml
  2  SET job = 'MANAGER'
  3  WHERE empno = 1001;
```

例 12-60 结果

```
已更新 1 行。
```

之后，可以使用例 12-61 的查询语句来验证您所做的修改是否正确。

例 12-61

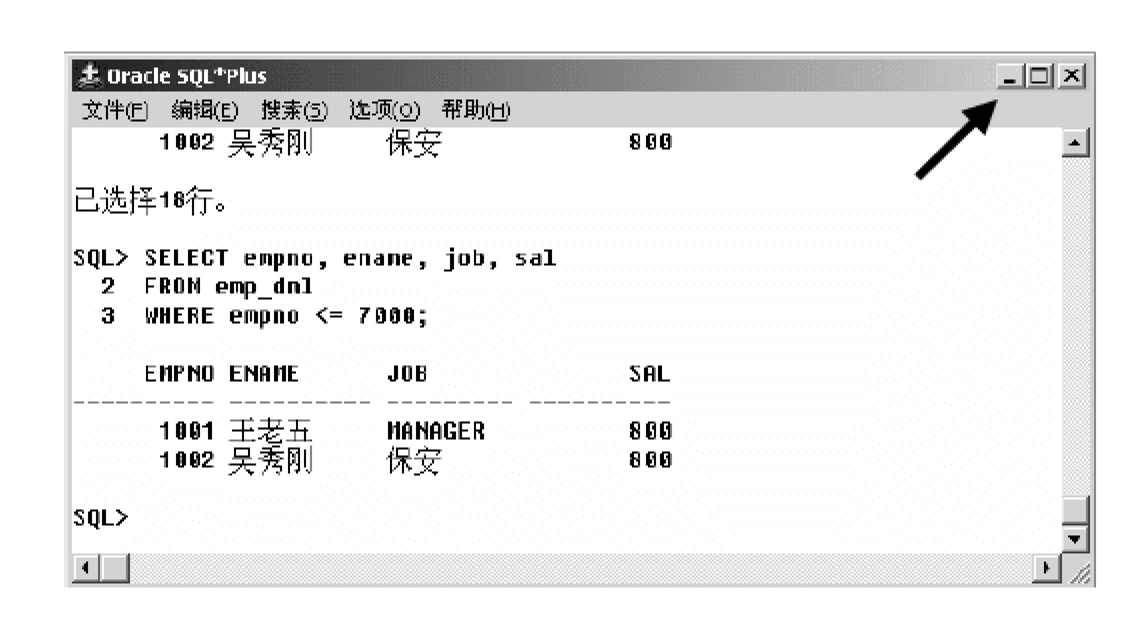
```
SQL> SELECT empno, ename, job, sal
  2  FROM emp_dml
  3  WHERE empno <= 7000;
```

例 12-61 结果

```
     EMPNO  ENAME        JOB              SAL
---------- ---------- ---------- ---------
      1001  王老五       MANAGER          800
      1002  吴秀刚       保安             800
```

例 12-61 的显示结果表明您的修改正确无误。这时经理找您有紧急的事情要处理，您可能用鼠标单击 SQL*Plus 窗口右上角的“关闭”按钮来退出 SQL*Plus，如例 12-62 所示。

例 12-62



等过了一会儿，您处理完了那件紧急的事情之后，又重新使用 SCOTT 用户名登录进入 Oracle 数据库中，之后，再次使用与例 12-61 相同的查询语句（例 12-63）来验证您所做的修改是否正确。

例 12-63

```
SQL> SELECT empno, ename, job, sal
   2  FROM emp_dml
   3  WHERE empno <= 7000;
```

例 12-63 结果

```
     EMPNO  ENAME         JOB             SAL
---------- ---------- ---------- ----------
      1001  王老五         保洁员          800
      1002  吴秀刚         保安            800
```

令您感到吃惊的是，此时王老五的职位（job）并不是经理（MANAGER），还是保洁员。这是因为您上面退出 SQL*Plus 的方式为非正常退出。在用户非正常退出 SQL*Plus 时，Oracle 会自动地回滚事务。

在解释了什么是非正常退出 SQL*Plus 之后，我们来解释什么是正常退出。现在您再一次使用与例 12-60 完全相同的 DML 语句（例 12-64）将王老五的职位（job）重新改为经理（MANAGER）。

例 12-64

```
SQL> UPDATE emp_dml
   2  SET job = 'MANAGER'
   3  WHERE empno = 1001;
```

例 12-64 结果

```
已更新 1 行。
```

这次您使用了 SQL*Plus 的 EXIT 命令退出 SQL*Plus，如例 12-65 所示。

例 12-65

```
SQL> exit
```

然后再以 SCOTT 用户重新登录到 Oracle 数据库中，现在可以再次使用与例 12-61 相同的查询语句（例 12-66）来验证您所做的修改是否确定已写到数据库中。

例 12-66

```
SQL> SELECT empno, ename, job, sal
  2  FROM emp_dml
  3  WHERE empno <= 7000;
```

例 12-66 结果

```
     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      1001 王老五     MANAGER          800
      1002 吴秀刚     保安              800
```

从例 12-66 的显示结果得知，您所做的修改已经写到数据库中，这是因为您用 SQL*Plus 的 EXIT 命令退出 SQL*Plus，是属于正常退出 SQL*Plus。

在这里我们并没有给出由系统崩溃结束事务的例子。其原因很简单，我并不想在自己的计算机上做这样的例子。

如果有足够的胆量想试试这样的例子，您可以在 SQL*Plus 中执行一些 DML 语句（不包括 DDL 或 DCL 语句，也不要使用 COMMIT 或 ROLLBACK 语句），之后重启计算机系统（不要正常关闭系统），当系统重启后您就会发现系统崩溃是如何结束事务了。希望您的运气不错，即这样的操作不至于损坏其他的系统文件。记住千万不要在生产系统上尝试这类的例子，除非您不想在公司或单位干了。

12.19　利用AUTOCOMMIT进行事务控制

像其他的数据库管理系统一样，您也可以让 Oracle 数据库管理系统每执行一条 DML 语句就提交一次事务。使用例 12-67 的命令将 Oracle 数据库管理系统设置成每执行一条 DML 语句就提交一次事务的状态。

例 12-67

```
SQL> SET AUTOCOMMIT ON
```

现在您可以使用例 12-68 的 DML 语句将工号为 1002 的员工晋升为经理(MANAGER)。

例 12-68

```
SQL> UPDATE emp_dml
  2  SET job = 'MANAGER'
  3  WHERE empno = 1002;
```

例 12-68 结果

```
已更新 1 行。
提交完成。
```

尽管例 12-68 结果已显示“提交完成”，但为了慎重起见，您可以利用另一个用户名登录数据库（如 SYSTEM/MANAGER）。之后，您还是应该使用例 12-69 的查询语句来验证一下您所做的修改是否确定被写到数据库中。

例 12-69

```
SQL> SELECT empno, ename, job, sal
  2  FROM scott.emp_dml
  3  WHERE empno <= 7000;
```

例 12-69 结果

```
     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      1001 王老五     MANAGER          800
      1002 吴秀刚     MANAGER          800
```

例 12-69 显示的结果表明您所做的修改已经被成功地写到数据库中。

将 AUTOCOMMIT 设成 ON，在进行 DML 操作时似乎挺方便，但是在实际应用中有时可能会出问题。例如，在有些应用中可能同时要对几个表进行 DML 操作，如果这些表已经利用外键建立了联系，那么由于外键约束的作用就使得 DML 操作与次序有关，因为 Oracle 数据库管理系统要维护引用完整性，这可能给应用程序的开发增加不少的困难，同时也提高了对程序水平的要求。好在 Oracle 数据库管理系统的默认设置 AUTOCOMMIT 是 OFF。

12.20 有关事务处理应注意的一些问题

如果读者学过数据库原理或读过类似的书，可能还记得大多数这类书都告诉读者：“在有人对数据库中的数据进行写操作（即 DML 操作）时，数据库系统为了维护数据的一致性不允许其他的人对该数据进行任何的操作（包括读操作）。数据库系统在进行写操作之前，先把要操作的数据行（记录）用排他锁（exclusive lock）锁上。只要记录上有排他锁，数据库系统就不允许其他的人对该数据行进行任何的操作。”

尽管在学术或理论上这样的设计似乎很完美，但在实际应用中常常会遇到困难。设想一下在一个每天 24 小时、每周 7 天运行的大型商业数据库中，某人修改了一个表中的数百条记录，而这些记录恰恰又是用户经常访问的数据。如果此人执行了 DML 语句之后既不

提交也不回滚，那么您可以想象出访问这些记录的用户的反应如何。

Oracle 数据库系统采取了另一种设计，即在有人对数据库中的数据进行写操作（即 DML 操作）时，Oracle 数据库系统允许其他的人对该数据进行读操作，这样就解决了上面所提到的问题。那么 Oracle 又是如何解决这一难题的呢？在每一个 Oracle 数据库中都有一个或多个称为回滚段的磁盘区（在 Oracle 9i 之前的版本称为 ROLLBACK SEGMENT，在 Oracle 9i 中称为 UNDO SEGMENT）。当有人对数据库中的数据进行任何写操作（即 DML 操作）时，Oracle 数据库系统首先将原始的数据复制到回滚段中，之后才做相应的操作，在事务处理结束之前其他的用户可以读这些数据，但读的是回滚段上的数据。

Oracle 数据库系统的这一设计确实大大提高了大型商业数据库的效率，但同时也会造成数据的不一致，因为在同一时刻，同一个数据可能会有两个不同的值，不过在大多数商业机构中是可以容忍的。例如，大公司在做年终结算（生成年报表）时差千八百块钱也无所谓。另外，公司可以用规章或规则来减缓这种数据的不一致，例如，公司可以规定在做年报表或月报表时不允许进行任何 DML 操作，公司也可以规定要在下班之后做年报表或月报表。

在 DML 操作之前将原始数据复制到回滚段中的设计本身在某些情况下也会产生效率方面的问题。例如，在一个大型的商业数据库中数据库操作员在维护时使用 DELETE 语句删除了一个一百万条记录的表。这样一个 DML 操作将要在回滚段上产生一百万条相同的记录项，这有可能会将回滚段所在的磁盘空间耗光，造成 Oracle 数据库系统的挂起。因此，如果要删除一个大表，为了数据库运行的效率，您可使用 TRUNCATE 语句而不用 DELETE 语句，因为 TRUNCATE 语句是 DDL 语句，不需要使用回滚段。

12.21　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- DML 语言包含了哪几个语句？
- 3 个 DML 语句的基本用法。
- 包含子查询的 DML 语句的用法。
- 在 DML 语句中如何处理空值（NULL）？
- 包含替代变量的 DML 语句的用法。
- 如何开发包含 DML 语句的脚本文件（Script）？
- 基于另一个表的 UPDATE 和 DELETE 操作。
- 在 UPDATE 和 DELETE 语句中 WHERE 子句的重要性。
- 为什么要引入事务（Transactions）处理？
- 如何标识一个事务（Transactions）？
- 用 COMMIT 和 ROLLBACK 显式结束事务（Transactions）。
- 隐式结束事务（Transactions）的方法及可能的隐患。
- 如何利用 AUTOCOMMIT 命令进行事务控制及其要注意的问题？
- Oracle 使用回滚段来管理事务处理的优缺点。

第13章 索引与约束

尽管一些有关 Oracle SQL 的书是将索引与约束放在不同的章中讨论，但由于这两者之间有许多联系，本书把它们放在同一章中讨论。

13.1 为什么引入索引

设想一下，在大型的商业数据库中有一个包含了数百万个记录的表，如果在这个表上没有索引的话，那么查询该表中的任何记录都只能通过顺序地逐行扫描得到，这会产生大量的磁盘输入/输出（I/O），因此也就会大大地降低系统的效率。

其实索引的概念很简单。许多读者可能有到图书馆借书的经历，当您到了图书馆之后，没有必要一个书架一个书架地看，一般您会先查看一下图书馆的图书目录，可以按书名、作者名或出版日期等查询。由于图书目录是按一定的顺序排放的，会很快地找到所需要的书的记录，在该记录中标有此书存放的准确位置，这时您就可以直奔存书的地方了。

Oracle 为了提高查询的效率也引入了索引（Indexes）。Oracle 索引也是按索引关键字的顺序存放记录，也称为数据结构。在索引记录中存有索引关键字和指向表中真正数据的指针（地址）。Oracle 系统利用算法在索引上可以很快地查找到所需的记录，并利用指针找到所需的数据。由于 Oracle 索引中只存索引关键字和指向表中真正数据的指针（地址），因此，它的规模要比真正存有数据的表的规模小得多。这样对索引进行操作的 I/O 量要比对真正的表进行操作少很多。读者可能知道在计算机的所有操作当中，I/O 操作应该是最慢的，因此，减少了 I/O 操作就等于加快了查询的速度。

引入索引的目的就是为了加快查询的速度。Oracle 索引是一个独立于表的对象，它可以存放在与表不同的磁盘上。即使索引崩溃，甚至索引被删除掉都不会影响真正存有数据的表。在 Oracle 数据库中，一个索引一旦被建立就由 Oracle 系统自动维护，而且由 Oracle 系统决定什么时候使用这个索引。您不用在查询语句中指定使用哪个索引，其实您所使用的查询语句与没建索引时几乎完全一样，只是查询速度可能快多了。虽然 Oracle 索引是一个独立于表的对象，但是当一个表被删除时所有基于该表的索引都被自动地删除掉。想想看为什么？

谈了这么多 Oracle 索引的内容，读者可能会问在 Oracle 数据库中怎样建立索引呢？

13.2　如何建立索引

在 Oracle 数据库中有如下两种方法来建立索引：

- Oracle 系统自动建立。当用户在一个表上建立主键（PRIMARY KEY）或唯一（UNIQUE）约束时，Oracle 系统会自动创建唯一索引（UNIQUE INDEX）。
- 手工建立。用户在一个表中的一列或多列上用 CREATE INDEX 语句来创建非唯一索引（NONUNIQUE INDEX）。

本节主要介绍手工建立索引，自动创建索引将在后面有关约束的章节中详细介绍。

创建索引的命令格式如下：

CREATE INDEX 索引名

ON 表名(列名[, 列名]...);

下面我们用例子来演示如何使用该命令手工创建索引。像以前一样，为了安全起见，我们先使用例 13-1 的 DDL 语句建立一个名为 empcon 的临时表。

例 13-1

```
SQL> CREATE TABLE empcon
  2  AS
  3  SELECT *
  4  FROM EMP;
```

例 13-1 结果

```
表已创建。
```

可以使用例 13-2 的 SQL*Plus 命令来验证是否已经成功创建了 empcon 表。

例 13-2

```
SQL> DESC empcon
```

例 13-2 结果

```
名称                                    是否为空? 类型
--------------------------------------- -------- ------------
EMPNO                                            NUMBER(4)
ENAME                                            VARCHAR2(10)
JOB                                              VARCHAR2(9)
MGR                                              NUMBER(4)
HIREDATE                                         DATE
SAL                                              NUMBER(7,2)
COMM                                             NUMBER(7,2)
DEPTNO                                           NUMBER(2)
```

还可以使用查询语句来检查 empcon 表中的数据是否正确。

假设用户经常按员工的名字（ename）查询，这时您可以使用例 13-3 的 DDL 语句在 ename 列上建立一个索引。

例 13-3

```
SQL> CREATE INDEX empcon_ename_idx
   2  ON empcon(ename);
```

例 13-3 结果

```
索引已创建。
```

注意：

在例 13-3 中索引名的命名方式是 Oracle 公司推荐的方式，其索引名包括了 3 部分，即表名、列名和对象的类型。以这种方式命名的索引将来维护起来很方便。您可以不遵守这一约定，使用像 aa、x1 等作为索引名，但将来维护起来很困难。

从例 13-3 的结果可知索引 empcon_ename_idx 已创建，您可能会问有没有办法看到这个索引呢？

13.3 如何查看索引

您可以使用 Oracle 的数据字典 user_indexes 来查看在您的用户下的索引信息。为了使显示的信息清楚易懂，可以先使用例 13-4 和例 13-5 的 SQL*Plus 命令来格式化 SELECT 语句的输出。

例 13-4

```
SQL> COL INDEX_TYPE FOR A10
```

例 13-5

```
SQL> COL TABLE_NAME FOR A10
```

现在就可以使用例 13-6 的查询语句来查看在您的用户下（SCOTT）所有的索引信息。

例 13-6

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME, UNIQUENESS
   2  FROM user_indexes;
```

例 13-6 结果

```
INDEX_NAME                     INDEX_TYPE TABLE_NAME UNIQUENES
------------------------------ ---------- ---------- ---------
EMPCON_ENAME_IDX               NORMAL     EMPCON     NONUNIQUE
PK_DEPT                        NORMAL     DEPT       UNIQUE
PK_EMP                         NORMAL     EMP        UNIQUE
SYS_C002718                    NORMAL     E_M_SHELL  UNIQUE
```

例 13-6 的结果告诉我们，在您的用户下共有 4 个索引。其中，索引 EMPCON_ENAME_

IDX 是基于 EMPCON 表的非唯一索引，它是正常类型的索引。最后一个索引是由 Oracle 系统命名的，它的名字为 SYS_C002718，该索引是基于 E_M_SHELL 表的唯一索引，也为正常类型的索引。从这一结果中您应该体会到 Oracle 公司推荐的索引名命名方式的好处。因为从您刚创建的索引的名字看，人们很容易想到这是一个基于 EMPCON 表中 ENAME 列的索引。

例 13-6 的结果并未告诉我们这些索引都是基于哪些列，要想得到这方面的信息，您可以使用数据字典 user_ind_columns。为了使显示的信息清楚易懂，您可以先使用例 13-7 的 SQL*Plus 命令来格式化 SELECT 语句的输出。

例 13-7

```
SQL> col column_name for a15
```

现在可以使用例 13-8 的查询语句来查看在您的用户下（SCOTT）所有与索引列有关的索引信息。

例 13-8

```
SQL> SELECT index_name, table_name, column_name, column_position
  2  FROM user_ind_columns;
```

例 13-8 结果

```
INDEX_NAME                     TABLE_NAME COLUMN_NAME     COLUMN_POSITION
------------------------------ ---------- --------------- --------
PK_DEPT                        DEPT       DEPTNO                         1
PK_EMP                         EMP        EMPNO                          1
EMPCON_ENAME_IDX               EMPCON     ENAME                          1
SYS_C002718                    E_M_SHELL  E_ID                           1
```

例 13-8 的结果不但告诉了我们这些索引基于哪些表，还告诉了我们它们基于哪些列。例如，索引 EMPCON_ENAME_IDX 是基于 EMPCON 表的 ENAME 列，而索引 PK_DEPT 是基于 DEPT 表的 DEPTNO 列等。您可能会对 COLUMN_POSITION 的含义感到困惑，请不用着急，通过下面的例子您会很容易地得知它的含义。首先使用例 13-9 的 DDL 语句在 job 和 sal 列上建立一个组合索引。

例 13-9

```
SQL> CREATE INDEX empcon_job_sal_idx
  2  ON empcon(job,sal);
```

例 13-9 结果

```
索引已创建。
```

现在再使用例 13-10 的查询语句来查看在您的用户下（SCOTT）所有与索引列有关的索引信息。

例 13-10

```
SQL> SELECT index_name, table_name, column_name, column_position
  2  FROM user_ind_columns;
```

例 13-10 结果

```
INDEX_NAME                     TABLE_NAME COLUMN_NAME     COLUMN_POSITION
------------------------------ ---------- ------------ ---------------
PK_DEPT                        DEPT        DEPTNO                        1
PK_EMP                         EMP         EMPNO                         1
EMP_ENAME_IDX                  EMPCON      ENAME                         1
EMPCON_JOB_SAL_IDX             EMPCON      JOB                           1
EMPCON_JOB_SAL_IDX             EMPCON      SAL                           2
SYS_C002718                    E_M_SHELL   E_ID                          1
已选择 6 行。
```

例 13-10 的结果表明，在组合索引 EMPCON_JOB_SAL_IDX 中最左边的索引关键字（列）JOB 的 COLUMN_POSITION 的值为 1，而组合索引 EMPCON_JOB_SAL_IDX 中左边第 2 位的索引关键字（列）SAL 的 COLUMN_POSITION 的值为 2。现在您应该理解 COLUMN_POSITION 的含义了吧？

13.4 使用索引时应注意的问题

尽管在一个表上建立索引可能会加快查询的速度，但这可能会降低 DML 操作的速度。因为每一条 DML 语句只要涉及索引关键字，Oracle 系统就要调整索引。另外，索引作为一个独立的对象需要消耗磁盘空间。如果表很大的话，其索引消耗磁盘空间量也会很大。

一些刚入门的数据库开发人员在刚刚学会建立索引时，往往趋向于在他们操作的表中的几乎每一列上都创建索引，这可能会消耗大量的磁盘空间，而且会大大地降低这些表的 DML 操作的速度。切记一定要避免在一个表上创建过多的索引（Over Indexes），尤其是对 DML 操作频繁的表。在建立每一个索引时，您都要权衡由这个索引所带来的查询速度的改进和它对 DML 操作的冲击。您一定要问自己：“对您的系统来说是查询操作重要呢？还是 DML 操作重要呢？”没有一个完美的答案，最后是在这两种操作之间作出折中。

另外一个值得注意的问题是，虽然您建立了索引，但 Oracle 系统并不保证一定使用它。为了让 Oracle 系统有可能使用索引，您应该把索引关键字放在 SELECT 语句的 WHERE 子句中。但这也不能保证 Oracle 系统百分之百地使用索引。例如，把组合索引中的一个索引关键字放在了 SELECT 语句的 WHERE 子句中，而该索引关键字不是最左边的索引关键字，此时 Oracle 系统将不使用该组合索引（Oracle 9i 在这方面做了一些改进）。另外，索引（列）关键字为 SELECT 语句中某个表达式的一部分时，Oracle 系统将可能不使用该索引。

还需要补充的一点是，尽管建立了索引，但 Oracle 9i 之前的版本对于这些索引是否被使用过，在 Oracle 系统中没有任何记录。Oracle 9i 对这方面作出了一些改进。Oracle 9i 可以记录某个索引是否被使用过，但只记录用过或没用过（YES/NO），即用过一次是用过，用过一万次也是用过。不管怎样 Oracle 9i 还是向前迈进了一步。

那么什么时候应该建立索引，什么时候不应该建立索引呢？对这一问题 Oracle 并没有给出一个精确的答案，而只给出了一些指导原则。

在下列条件之一成立时，您应该为这个表建立索引：

- 表很大而且大多数查询的返回数据量很少（Oracle 推荐为小于总行数的百分之二到百分之四）。因为如果返回数据量很大的话就不如顺序地扫描这个表了。
- 此列的取值范围很广，一般为随机分布。如在大多数员工表中的年龄（age）列一般为随机分布，即几乎从 18 岁到 60 岁（许多西方国家为 65 岁）所有年龄的员工都有（但在军队和夜总会员工的年龄可能就主要集中在 18 到 20 岁之间，这样的数据就不是随机分布）。
- 一列或多列经常出现在 WHERE 子句或连接条件中，原因在前面已经解释了。
- 表上的 DML 操作较少。
- 此列中包含了大量的空值（NULL）。
- 此列不经常作为 SELECT 语句中某个表达式的一部分。

将以上的陈述反过来说就是什么时候不应该建立索引。

13.5　基于函数的索引

如果在 SELECT 语句的 WHERE 子句中使用了某一表达式，如何加快这样的查询速度呢？在 Oracle 8i 以前的版本上是没有办法做到的，但在 Oracle 8i 或以后的版本上可以利用创建基于函数的索引的方法来解决这一难题。与一般的索引不同的是，基于函数的索引中的索引关键字是表达式而不是列。

假如在您的公司中，工资在 2000 元以下为低收入者。您或者经理们可能时常要查看一下，哪些员工属于低收入者，哪些员工不属于低收入者。为了加快这类查询的速度，您试着用例 13-11 的 DDL 语句来建立一个基于表达式 sal−2000 的索引。

例 13-11

```
SQL> CREATE INDEX empcon_salgt_idx
  2  ON empcon(sal-2000);
```

例 13-11 结果

```
ON empcon(sal-2000)
           *
ERROR 位于第 2 行:
ORA-01031: 权限不足
```

例 13-11 显示的结果可能使您感到困惑。这是因为您是以普通用户登录的，所以权限不够（我们很快会解释其中的原因）。现在您可以使用例 13-12 的 SQL*Plus 命令用 DBA 用户重新登录。

例 13-12

```
SQL> connect system/manager
```

例 13-12 结果

```
已连接。
```

之后，可以重新使用与例 13-11 完全相同的 SQL*Plus 命令，利用例 13-13 来创建所需的基于表达式 sal−2000 的索引。

例 13-13

```
SQL> CREATE INDEX empcon_salgt_idx
  2  ON scott.empcon(sal-2000);
```

例 13-13 结果

```
索引已创建。
```

尽管例 13-13 结果已显示“索引已创建。”，但为了慎重起见还是应该使用查询语句来验证是否真正成功地创建这个索引。为了使显示的信息清楚易懂，应该先使用例 13-14～例 13-16 的 SQL*Plus 命令来格式化 SELECT 语句的输出。

例 13-14

```
SQL> COL INDEX_TYPE FOR A10
```

例 13-15

```
SQL> COL INDEX_TYPE FOR A25
```

例 13-16

```
SQL> COL index_name for a20
```

现在再使用例 13-17 的查询语句来查看刚创建的索引信息。

例 13-17

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME, UNIQUENESS
  2  FROM user_indexes
  3  WHERE table_owner = 'SCOTT';
```

例 13-17 结果

```
INDEX_NAME           INDEX_TYPE                TABLE_NAME UNIQUENES
-------------------- ----------------------- ---------- ---------
EMPCON_SALGT_IDX     FUNCTION-BASED NORMAL     EMPCON     NONUNIQUE
```

请注意，例 13-17 显示的结果中 INDEX_TYPE 已变成 FUNCTION-BASED NORMAL 了。现在您再使用例 13-18 的查询语句来查看在刚创建的索引中与索引列有关的信息。

例 13-18

```
SQL> SELECT index_name, table_name, column_name, column_position
  2  FROM user_ind_columns
  3  WHERE table_name = 'EMPCON';
```

例 13-18 结果

```
INDEX_NAME           TABLE_NAME COLUMN_NAME     COLUMN_POSITION
-------------------- ---------- --------------- ---------------
EMPCON_SALGT_IDX     EMPCON     SYS_NC00009$                  1
```

从例 13-18 显示的结果中可以看到 COLUMN_NAME 为 SYS_NC00009$，这是 Oracle 系统为表达式 sal−2000 自动生成的列名。

13.6　如何确认Oracle系统是否使用了索引

虽然在 Oracle 数据库中，由 Oracle 系统决定什么时候使用索引，不用在查询语句中指定使用哪个索引。但是 Oracle 系统究竟有没有使用索引一直是一个扑朔迷离的问题。可能您想知道到底有没有办法确切地知道您所建立的索引是否真的使用过，可以使用 SQL*Plus 的 EXPLAIN 命令来得到有关索引使用情况的信息。但在使用这个命令之前，您需要运行类似例 13-19 的 utlxplan.sql 脚本文件来产生 plan_table 表。

例 13-19

```
@d:\Oracle\ora90\rdbms\admin\utlxplan
```

在例 13-19 中，d:\Oracle\ora90 为 Oracle 的安装目录，也就是在有些书中称之为 $Oracle_HOME 的目录。在您的 Oracle 系统上可能不同。如果您找不到这个脚本文件，可用搜索器（在许多操作系统上为 find 命令）搜一下就行了。

之后，可以使用例 13-20 的 SQL*Plus 命令来解释您的查询语句。

例 13-20

```
SQL> EXPLAIN plan for
  2  SELECT ename, job, sal, comm, deptno
  3  FROM empcon
  4  WHERE (sal-2000) < 0;
```

例 13-20 结果

```
已解释。
```

为了使显示的信息清楚易懂，您应该先使用从例 13-21~例 13-24 的 SQL*Plus 命令来格式化 SELECT 语句的输出。

例 13-21

```
SQL> col id for 999
```

例 13-22

```
SQL> col operation for a20
```

例 13-23

```
SQL> col options for a15
```

例 13-24

```
SQL> col object_name for a18
```

现在就可以通过例 13-25 查询 plan_table 表，查看 Oracle 系统是否使用了您所创建的

empcon_salgt_idx 索引。

例 13-25

```
SQL> SELECT id, operation, options, object_name, position
   2  FROM plan_table;
```

例 13-25 结果

```
ID OPERATION            OPTIONS    OBJECT_NAM  POSITION
--- -------------------- ---------- ---------- ----------
  1 TABLE ACCESS          FULL       EMPCON            1
  0 SELECT STATEMENT
```

例 13-25 显示的结果表明，Oracle 系统并没有使用您所创建的 empcon_salgt_idx 索引，而是顺序地扫描了整个 EMPCON 表（Full Table Scan），这个结果多少有点令人失望。

现在您可以再检查一下 Oracle 系统是否使用了您所创建的另一个索引 empcon_ename_idx，为此，您应该先使用例 13-26 的 DDL 语句清除 plan_table 表中的内容。

例 13-26

```
SQL> truncate  table  plan_table;
```

例 13-26 结果

```
表已截掉。
```

之后，可以使用例 13-27 的 SQL*Plus 命令来解释您的查询语句。

例 13-27

```
SQL> EXPLAIN plan for
   2  SELECT empno, ename, job, sal
   3  FROM empcon
   4  WHERE ename LIKE 'J%';
```

例 13-27 结果

```
已解释。
```

现在就可以通过例 13-28 查询 plan_table 表，看看 Oracle 系统是否使用了您所创建的 empcon_ename_idx 索引。

例 13-28

```
SQL> SELECT id, operation, options, object_name, position
   2  FROM plan_table;
```

例 13-28 结果

```
  ID OPERATION            OPTIONS         OBJECT_NAME          POSITION
---- -------------------- --------------- -------------------- ------
   0 SELECT STATEMENT
   1 TABLE ACCESS          BY INDEX ROWID  EMPCON                     1
   2 INDEX                RANGE SCAN      EMPCON_ENAME_IDX            1
```

例 13-28 显示的结果表明，Oracle 系统使用了您所创建的索引 EMPCON_ENAME_IDX，这一点多少令您感到欣慰。

以上的例子表明，Oracle 系统什么时候使用索引往往会出乎人们的预料。

注意：

本节的内容并不要求读者掌握，了解即可。

13.7　如何删除索引

当一个索引不再需要时，应该删除它以释放这个索引所占有的磁盘空间。另外，在大规模输入之前，为了加快输入数据的速度有时也应该先删除索引，等数据输入之后再重建这些索引。删除索引的命令格式如下：

DROP INDEX 索引名;

该 DDL 命令将从数据字典中删除索引的定义，并释放这个索引所占用的磁盘空间。要删除一个索引，您必须是索引的所有者或是具有 DROP ANY INDEX 的系统权限。

假设您发现组合索引 empcon_job_sal_idx 没多大的用处，可以使用例 13-29 的 DDL 语句删除这个索引。

例 13-29

```
SQL> DROP INDEX EMPCON_JOB_SAL_IDX;
```

例 13-29 结果

```
索引已丢弃。
```

尽管例 13-29 的结果已显示“索引已丢弃”，但为了慎重起见，您还想验证一下组合索引 empcon_job_sal_idx 是否真的被删除了。为了使显示的信息清楚易懂，还应该先使用例 13-30 和例 13-31 的 SQL*Plus 命令来格式化 SELECT 语句的输出。

例 13-30

```
SQL> COL INDEX_TYPE FOR A10
```

例 13-31

```
SQL> COL TABLE_NAME FOR A10
```

现在可以使用例 13-32 的查询语句来查看在您的用户下所有的索引信息。

例 13-32

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME, UNIQUENESS
  2  FROM user_indexes;
```

例 13-32 结果

```
INDEX_NAME                                        INDEX_TYPE TABLE_NAME  UNIQUENES
------------------------------------------------ ------------ ----------- ------------
```

```
EMPCON_ENAME_IDX                          NORMAL     EMPCON        NONUNIQUE
PK_DEPT                                   NORMAL     DEPT          UNIQUE
PK_EMP                                    NORMAL     EMP           UNIQUE
SYS_C002718                               NORMAL     E_M_SHELL     UNIQUE
```

例 13-32 显示的结果表明，您已经成功删除了组合索引 empcon_job_sal_idx。

假设您是一个数据库管理员（DBA），现在需要向 EMPCON 表中输入大量的数据，为了加快输入数据的速度，您应该先删除 EMPCON 表上的非唯一的索引 EMPCON_ENAME_IDX。等数据输入之后，如果需要的话，可再重建这个索引。

为了模拟以上所说的情形，可以使用例 13-33 的命令以 DBA 用户重新登录。

例 13-33

```
SQL> CONNECT SYSTEM/MANAGER
```

例 13-33 结果

```
已连接。
```

现在您就可以使用例 13-34 的 DDL 语句删除 SCOTT 用户的索引 EMPCON_ENAME_IDX。

例 13-34

```
SQL> DROP INDEX SCOTT.EMPCON_ENAME_IDX
```

例 13-34 结果

```
索引已丢弃。
```

现在您可以使用与例 13-32 相似的查询语句例 13-35 来查看在 SCOTT 用户下所有的索引信息。

例 13-35

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME, UNIQUENESS
   2  FROM dba_indexes
   3  WHERE owner = 'SCOTT';
```

例 13-35 结果

```
INDEX_NAME                     INDEX_TYPE TABLE_NAME UNIQUENES
------------------------------ ---------- ---------- ---------
PK_DEPT                           NORMAL     DEPT          UNIQUE
PK_EMP                            NORMAL     EMP           UNIQUE
SYS_C002718                       NORMAL     E_M_SHELL     UNIQUE
```

例 13-35 的显示结果表明，您已经成功删除了基于表 EMPCON 的索引 EMPCON_ENAME_IDX。

如果您删除了一个表，那么所有基于该表的索引都会被自动地删除。

假设您仍在 SYSTEM 用户下，可以用例 13-36~例 13-38 的一些命令来显示所有基于表 EMPCON 的索引信息。

例 13-36

```
SQL> COL INDEX_NAME FOR A18
```

例 13-37

```
SQL> COL INDEX_TYPE FOR A22
```

例 13-38

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME, UNIQUENESS
  2  FROM dba_indexes
  3  WHERE table_name = 'EMPCON';
```

例 13-38 结果

```
INDEX_NAME         INDEX_TYPE                  TABLE_NAME UNIQUENES
------------------ ---------------------- ---------- ---------
EMPCON_SALGT_IDX   FUNCTION-BASED NORMAL  EMPCON     NONUNIQUE
```

例 13-38 的显示结果表明，基于 EMPCON 表有一个索引叫 EMPCON_SALGT_IDX。现在您就可以使用如下的 DDL 语句（例 13-39）删除 SCOTT 用户的表 EMPCON。

例 13-39

```
SQL> DROP TABLE scott.empcon;
```

例 13-39 结果

```
表已丢弃。
```

现在您再使用与例 13-38 完全一样的查询语句例 13-40 来显示所有基于表 EMPCON 的索引信息。

例 13-40

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME, UNIQUENESS
  2  FROM dba_indexes
  3  WHERE table_name = 'EMPCON';
```

例 13-40 结果

```
未选定行
```

例 13-40 的显示结果表明，随着 EMPCON 表的删除，所有基于表 EMPCON 的索引也都被自动地删除了。

13.8　为什么要引入约束及如何定义约束

引入约束（Constraints）的目的就是防止那些无效或有问题的数据输入到表中，使用数据库的术语就是维护数据的一致性。约束是强加在表上的规则或条件。当对该表进行 DML 或 DDL 操作时，如果此操作会造成表中的数据违反约束条件或规则的话，Oracle 系统就会拒绝执行这个操作。这样做的好处是，当错误刚一出现时就能被 Oracle 系统自动地发现，

从而使数据库的开发和维护都更加容易。

Oracle 系统共提供了以下 5 种约束。

- 非空（NOT NULL）约束：所定义的列决不能为空。
- 唯一（UNIQUE）约束：在表中每一行中所定义的这列或这些列的值都不能相同。
- 主键（PRIMARY KEY）约束：唯一地标识表中的每一行。
- 外键（FOREIGN KEY）约束：用来维护从表（Child Table）和主表（Parent Table）之间的引用完整性（Referential Integrity）。
- 条件（CHECK）约束：表中每行都要满足该约束条件。

约束是加在表上的，因为只有表中存有数据。可以在创建表时在 CREATE TABLE 语句中定义约束，也可以在已存在的表上利用 ALTER TABLE 语句来定义约束。既可以在列一级也可以在表一级定义约束。约束的定义存在 Oracle 的数据字典中，只能通过数据字典来浏览约束。可以给出约束的名字，如果您在定义约束时没有给出约束的名字，Oracle 系统将为该约束自动生成一个名字，其格式为 SYS_Cn，其中 n 为大于 0 的自然数。

13.9 非空约束

一般在一个公司或机构中，所有的部门都应该有一个部门名，也就是说，如果公司或机构要成立一个部门，就必须给出这个部门名，即部门名不能为空。现在您可以通过使用例 13-41 的 CREATE TABLE 语句来定义非空（NOT NULL）约束，以这种方法来实现上面所说的商业规则。

例 13-41

```
SQL> CREATE TABLE deptcon(
  2          deptno     NUMBER(3),
  3          dname      VARCHAR2(15)   NOT NULL,
  4          loc        VARCHAR2(20));
```

例 13-41 结果

```
表已创建。
```

现在您可以使用例 13-42 的 SQL*Plus 命令来检查是否已经成功地在 deptcon 表的 dname 列上定义了非空（NOT NULL）约束。

例 13-42

```
SQL> DESC deptcon
```

例 13-42 结果

```
名称                                       是否为空? 类型
 ----------------------------------------- -------- ------------
 DEPTNO                                             NUMBER(3)
 DNAME                                     NOT NULL VARCHAR2(15)
 LOC                                                VARCHAR2(20)
```

例 13-42 的显示结果表明，您已成功地在 deptcon 表的 dname 列上定义了非空（NOT NULL）约束。

在 Oracle 系统提供的 5 种约束中，非空（NOT NULL）约束是唯一一种只能在列一级定义的约束。现在您可以通过对 deptcon 表的 DML 操作来进一步理解非空（NOT NULL）约束的含义。首先使用例 13-43 的 DML 语句向 deptcon 表中插入一行正常的数据。

例 13-43

```
SQL> INSERT INTO deptcon(deptno, dname, loc)
  2  VALUES            (10, 'ACCOUNTING', 'BEIJING');
```

例 13-43 结果

```
已创建 1 行。
```

例 13-43 的显示结果表明，已经把一行数据成功地插入到 deptcon 表中，因为该行数据中部门名不为空，它满足在此列上所定义的约束。

现在再使用例 13-44 的 DML 语句向 deptcon 表中插入一行非正常（部门名为空）的数据。

例 13-44

```
SQL> INSERT INTO deptcon(deptno, dname, loc)
  2  VALUES            (20, '' , 'GUANGZGOU');
```

例 13-44 结果

```
INSERT INTO deptcon(deptno, dname, loc)
*
ERROR 位于第 1 行:
```

例 13-44 的显示结果表明，Oracle 系统拒绝执行插入操作，因为该行数据中部门名为空，它不满足在此列上所定义的约束。如果将该行数据中部门名改为 UNKNOWN 或任何非空的字符串，Oracle 系统就会执行插入操作，如例 13-45 所示。

例 13-45

```
SQL> INSERT INTO deptcon(deptno, dname, loc)
  2  VALUES            (20, 'UNKNOWN', 'GUANGZGOU');
```

例 13-45 结果

```
已创建 1 行。
```

现在可以使用例 13-46 的查询语句来查看 deptcon 中的内容。

例 13-46

```
SQL> SELECT *
  2  FROM deptcon;
```

例 13-46 结果

```
    DEPTNO DNAME          LOC
---------- -------------- ---------
```

```
        10 ACCOUNTING       BEIJING
        20 UNKNOWN         GUANGZGOU
```

测试完了 INSERT 语句，就该测试 UPDATE 语句了，可以使用例 13-47 的 DML 语句试着将第 20 号部门的名字改为空。

例 13-47

```
SQL> UPDATE deptcon
  2  SET dname = NULL
  3  WHERE deptno = 20;
```

例 13-47 结果

```
UPDATE deptcon
*
ERROR 位于第 1 行:
ORA-01407: 无法更新 ("SCOTT"."DEPTCON"."DNAME") 为 NULL
```

例 13-47 的显示结果表明，Oracle 系统拒绝执行修改操作，因为在该行数据中修改后的部门名为空，它不满足在此列上所定义的约束。

如果现在使用与例 13-47 相似如例 13-48 的 UPDATE 语句将第 20 号部门的名字改为 ACCOUNTING，Oracle 系统又该如何处理呢？

例 13-48

```
SQL> UPDATE deptcon
  2  SET dname = 'ACCOUNTING'
  3  WHERE deptno = 20;
```

例 13-48 结果

```
已更新 1 行。
```

例 13-48 的显示结果表明，您已经成功地把 deptcon 表中的第 20 号部门的名字改为 ACCOUNTING，因为该行数据中部门名不为空，它满足在此列上所定义的约束。

现在您可以使用例 13-49 的查询语句来查看 deptcon 中的内容。

例 13-49

```
SQL> SELECT *
  2  FROM deptcon;
```

例 13-49 结果

```
   DEPTNO DNAME          LOC
---------- -------------- ---------
        10 ACCOUNTING     BEIJING
        20 ACCOUNTING     GUANGZGOU
```

例 13-49 的显示结果表明，在 deptcon 表中有两个叫 ACCOUNTING 的部门。在实际的商业运作中这几乎是不可能的，但 Oracle 系统不管这些，因为这并未违反您在 DNAME 列

上定义的非空约束。您如果不想让 deptcon 表中的部门重复，就需要再加上唯一（UNIQUE）约束，所谓“铁路警察各管一段”就是这个意思。

您不用测试 DELETE 语句，因为该语句不会产生数据违反非空（NOT NULL）约束的情形。

13.10　查看有关约束的信息

对于非空（NOT NULL）约束，您有时可以使用 SQL*Plus 的 DESC 命令来得到一些有关这种约束的信息。但如果您想知道约束的名字，或者您想查看约束是不是非空（NOT NULL）约束，又该怎么办呢？

您可以使用数据字典 USER_CONSTRAINTS 来得到这些信息。

为什么要使用例 13-50 和例 13-51 的 SQL*Plus 命令现在您应该比较清楚了，在这里就不再解释了。

例 13-50

```
SQL> COL owner FOR A10
```

例 13-51

```
SQL> COL table_name FOR A10
```

之后，您可以使用例 13-52 的查询语句来查看 SCOTT 用户所创建的所有的约束。

例 13-52

```
SQL> SELECT owner, constraint_name, constraint_type, table_name
  2  FROM user_constraints;
```

例 13-52 结果

```
OWNER      CONSTRAINT_NAME                C TABLE_NAME
---------- ------------------------------ - ----------
SCOTT      PK_DEPT                        P DEPT
SCOTT      SYS_C002719                    C DEPTCON
SCOTT      PK_EMP                         P EMP
SCOTT      FK_DEPTNO                      R EMP
SCOTT      SYS_C002718                    P E_M_SHELL
SCOTT      SYS_C002715                    C MANAGER
SCOTT      SYS_C002716                    C MANAGER
SCOTT      SYS_C002717                    C MANAGER
已选择 8 行。
```

例 13-52 的显示结果表明，在 deptcon 表上定义了一个约束，约束的类型为 C，它的名字是 Oracle 系统自动生成的，为 SYS_C002719。

现在我们来解释例 13-52 的显示结果中第 3 列 C（constraint_type）中每个字母所代表

的含义。

- C：代表 CHECK（条件约束）和 NOT NULL（非空约束）。
- P：代表 PRIMARY KEY（主键约束）。
- R：代表 REFERENTIAL INTEGRITY，即 FOREIGN KEY（外键约束）。
- U：代表 UNIQUE（唯一约束）。

如果您想知道约束是定义在哪一个表的哪一列上的话，又该怎么办呢？您可以使用数据字典 USER_CONS_COLUMNS 来得到这方面的信息。现在您可以使用例 13-54 的查询语句来查看 SCOTT 用户所创建的所有约束的有关信息。

例 13-53

```
SQL> COL column_name for a15
```

例 13-54

```
SQL> SELECT owner, constraint_name, table_name, column_name
   2  FROM user_cons_columns;
```

例 13-54 结果

```
OWNER      CONSTRAINT_NAME                TABLE_NAME  COLUMN_NAME
---------- ------------------------------ -------- -----------
SCOTT      FK_DEPTNO                      EMP         DEPTNO
SCOTT      PK_DEPT                        DEPT        DEPTNO
SCOTT      PK_EMP                         EMP         EMPNO
SCOTT      SYS_C002715                    MANAGER     EMPNO
SCOTT      SYS_C002716                    MANAGER     ENAME
SCOTT      SYS_C002717                    MANAGER     HIREDATE
SCOTT      SYS_C002718                    E_M_SHELL   E_ID
SCOTT      SYS_C002719                    DEPTCON     DNAME
已选择 8 行。
```

例 13-54 的显示结果表明，在 deptcon 表上定义了一个约束，该约束是定义在 DNAME 上的，它的名字是 Oracle 系统自动生成的，为 SYS_C002719。

13.11 唯一约束

如果您的公司要求所有部门的名字都不能相同，那又该怎么办呢？您可以利用在 DNAME 上定义唯一（UNIQUE）约束来实现公司的这一商业规则。

为了操作方便，可以先使用例 13-55 的 DDL 语句将加在 deptcon 表的 DNAME 上的 NOT NULL（非空约束）删除。

例 13-55

```
SQL> ALTER TABLE deptcon
   2  DROP CONSTRAINT SYS_C002719;
```

例 13-55 结果

```
表已更改。
```

现在应该使用与例 13-54 的查询语句完全一样的查询语句例 13-56 来查看您所在的用户所创建的所有约束信息。

例 13-56

```
SQL> SELECT owner, constraint_name, table_name, column_name
  2  FROM user_cons_columns;
```

例 13-56 结果

```
OWNER     CONSTRAINT_NAME              TABLE_NAME COLUMN_NAME
--------- ---------------------------- ---------- -----------
SCOTT     FK_DEPTNO                    EMP         DEPTNO
SCOTT     PK_DEPT                      DEPT        DEPTNO
SCOTT     PK_EMP                       EMP         EMPNO
SCOTT     SYS_C002715                  MANAGER     EMPNO
SCOTT     SYS_C002716                  MANAGER     ENAME
SCOTT     SYS_C002717                  MANAGER     HIREDATE
SCOTT     SYS_C002718                  E_M_SHELL   E_ID
已选择 7 行。
```

例 13-56 的显示结果表明，您已经成功地删除了在 deptcon 表上所定义的全部约束。

现在可以试着使用例 13-57 的 DDL 语句在 deptcon 表的 dname 上添加一个唯一（UNIQUE）约束。

例 13-57

```
SQL> ALTER TABLE deptcon
  2  ADD CONSTRAINT deptcon_dname_uk UNIQUE(dname);
```

例 13-57 结果

```
ADD CONSTRAINT deptcon_dname_uk UNIQUE(dname)
             *
ERROR 位于第 2 行:
ORA-02299: 无法验证 (SCOTT.DEPTCON_DNAME_UK) - 未找到重复关键字
```

例 13-57 的显示输出可能令您感到困惑。不过用不着担心，您可以使用例 13-58 的查询语句看看到底发生了什么。

例 13-58

```
SQL> SELECT * FROM DEPTCON;
```

例 13-58 结果

```
   DEPTNO DNAME          LOC
--------- ------------- ---------
```

```
        10  ACCOUNTING        BEIJING
        20  ACCOUNTING        GUANGZGOU
```

看到例 13-58 的结果，您立刻明白了，这是因为在 deptcon 表中有两个叫 ACCOUNTING 的部门，毫无疑问这违反了唯一（UNIQUE）约束条件。现在您使用例 13-59 的 DML 语句可以删除第 2 行部门名为 ACCOUNTING 的记录。

例 13-59

```
SQL> DELETE FROM deptcon
  2  WHERE deptno = 20;
```

例 13-59 结果

```
已删除 1 行。
```

当看到例 13-59 的结果“已删除 1 行”后，就可以再使用与例 13-57 完全相同的 DDL 语句例 13-60 在 deptcon 表的 dname 上添加一个唯一（UNIQUE）约束。

例 13-60

```
SQL> ALTER TABLE deptcon
  2  ADD CONSTRAINT deptcon_dname_uk UNIQUE(dname);
```

例 13-60 结果

```
表已更改。
```

例 13-60 的显示结果表明，您已成功地在 deptcon 表的 dname 列上定义了唯一（UNIQUE）约束。为了慎重起见，应该使用例 13-61~例 13-63 的语句来查看您所在的用户所创建的所有约束信息。

例 13-61

```
SQL> COL SEARCH_CONDITION FOR A23
```

例 13-62

```
SQL> COL CONSTRAINT_NAME FOR A20
```

例 13-63

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         search_condition
  3  FROM user_constraints;
```

例 13-63 结果

```
OWNER       CONSTRAINT_NAME       C TABLE_NAME SEARCH_CONDITION
----------- --------------------- - ---------- -------------------
SCOTT       PK_DEPT               P DEPT
SCOTT       DEPTCON_DNAME_UK      U DEPTCON
SCOTT       PK_EMP                P EMP
SCOTT       FK_DEPTNO             R EMP
SCOTT       SYS_C002718           P E_M_SHELL
```

```
SCOTT       SYS_C002715              C MANAGER    "EMPNO" IS NOT NULL
SCOTT       SYS_C002716              C MANAGER    "ENAME" IS NOT NULL
SCOTT       SYS_C002717              C MANAGER    "HIREDATE" IS NOT NULL
已选择 8 行。
```

例 13-63 的显示结果表明，在 deptcon 表上定义了一个约束，该约束的类型为 UNIQUE，它的名字是 DEPTCON_ DNAME_UK。

其实在这个例子中，只要看到了约束名 DEPTCON_DNAME_UK，您就应该猜到该约束是定义在 DEPTCON 表的 DNAME 列上，并且是一个唯一（UNIQUE）约束。这种约束的命名法是 Oracle 推荐的，它由 3 部分组成，即表名、列名和约束的类型，它们之间由下划线（_）连接。其中约束的类型表示如下。

- UK：UNIQUE KEY（唯一）约束。
- PK：PRIMARY KEY（主键）约束。
- FK：FOREIGN KEY（外键）约束。
- CK：CHECK（条件）约束。
- NN：NOT NULL（非空）约束。

例 13-57 已经验证了唯一（UNIQUE）约束不允许表中有两行相同的数据，那么 UNIQUE KEY（唯一）约束允许插入空值（NULL）吗？我想最好的办法是用例子来说明。首先您可以试着使用例 13-64 的 DML 语句向 deptcon 表中插入一行部门名为空的数据。

例 13-64

```
SQL> INSERT INTO deptcon(deptno, dname, loc)
  2  VALUES            (20, NULL, '牛街');
```

例 13-64 结果

```
已创建 1 行。
```

例 13-64 的显示结果表明，UNIQUE KEY（唯一）约束允许插入一个空值（NULL）。现在您可能又要问 UNIQUE KEY（唯一）约束允许插入多个空值（NULL）吗？为了回答这个问题，我们还是使用老办法用例子来说明。您可以再试着使用与例 13-64 相似如例 13-65 的 DML 语句向 deptcon 表中插入一行部门名为空的数据。

例 13-65

```
SQL> INSERT INTO deptcon(deptno, dname, loc)
  2  VALUES            (20, NULL, '狼山镇');
```

例 13-65 结果

```
已创建 1 行。
```

例 13-65 的显示结果表明，UNIQUE KEY（唯一）约束允许插入第 2 个空值（NULL）。其实，UNIQUE KEY（唯一）约束允许插入任意多个空值（NULL）。因为空值（NULL）不等于任何值，所以每个空值（NULL）都不等于任何其他的空值（NULL），即每个空值（NULL）都是唯一的。您可以重温第 5 章中所讲的那个世俗的故事。假设有 N 位白马王

子向那位美丽动人的少女立下了同样的爱的誓言，谁能说出他们的誓言是相等还是不等呢？我相信没人能回答这个问题。

我们并未给出利用 UPDATE 语句来测试 UNIQUE KEY（唯一）约束的例子，如果读者感兴趣可以自己试着做一下。

与非空（NOT NULL）约束一样，您不用测试 DELETE 语句，因为该语句不会产生数据违反 UNIQUE KEY（唯一）约束的情形。

13.12 条件约束

您可以利用条件（CHECK）约束来实现公司中一些比较复杂的商业规则。条件（CHECK）约束定义了表中每一行数据都必须满足的条件。条件（CHECK）约束中的条件与查询语句中的条件相同，但是不能包括以下的内容：

- CURRVAL、NEXTVAL、LEVEL 和 ROWNUM 这样的伪列（PSEUDOCOLUMNS）。
- 引用其他行中值的查询语句。
- SYSDATE、USER、USERENV 和 UID 的函数调用。

条件（CHECK）约束既可以在表一级定义也可以在列一级定义，在一列上可以定义任意多个条件（CHECK）约束。

为了使读者容易理解，还是老办法用例子来说明。如果您留意过招工广告的话，您可能有印象，一般对前台或文秘的招工都要求应征者满足以下条件：

- 女性。
- 年龄在 18~35 岁之间。
- 最好大学或以上学历。
- 相貌端庄。
- 未婚等。

您可以使用例 13-66 的 DDL 语句来实现上述的第 1 和第 2 个条件（有些条件很难在计算机上实现，如相貌端庄）。

例 13-66

```
SQL> CREATE TABLE person(
           id      VARCHAR2(10),
           name    VARCHAR2(20),
           gender  CHAR(1),
           age     NUMBER,
           CONSTRAINT person_gender_ck
                   CHECK(gender = 'F'),
           CONSTRAINT person_age_ck
                   CHECK(age BETWEEN 18 AND 35));
```

例 13-66 结果

```
表已创建。
```

虽然您看到了例 13-66 结果“表已创建”，但作为一名老练的 Oracle 数据库管理员（DBA），您还是应该使用例 13-67 的 SQL*Plus 命令来检查您是否已经成功地创建了表 person。

例 13-67

```
SQL> DESC person
```

例 13-67 结果

```
名称                                          是否为空？ 类型
--------------------------------------------- --------- ------------
 ID                                                     VARCHAR2(10)
 NAME                                                   VARCHAR2(20)
 GENDER                                                 CHAR(1)
 AGE                                                    NUMBER
```

当看到了例 13-66 结果之后，您的心里一定踏实多了。接下来，还是应该使用例 13-72 的查询语句来检查是否已经成功地在 person 表上定义了您所需要的条件（CHECK）约束。例 13-68~例 13-71 的作用您应该比较清楚了，这里就不再解释了。

例 13-68

```
SQL> COL owner FOR A8
```

例 13-69

```
SQL> COL CONSTRAINT_NAME FOR A20
```

例 13-70

```
SQL> COL TABLE_NAME FOR A10
```

例 13-71

```
SQL> COL SEARCH_CONDITION FOR A25
```

例 13-72

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         search_condition
  3  FROM user_constraints
  4  WHERE table_name = 'PERSON';
```

例 13-72 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME SEARCH_CONDITION
-------- -------------------- - ---------- -------------------------
SCOTT    PERSON_GENDER_CK     C PERSON     gender = 'F'
SCOTT    PERSON_AGE_CK        C PERSON     age BETWEEN 18 AND 35
```

例 13-72 的显示结果表明，在 SCOTT 用户的 PERSON 表上定义了两个约束，第 1 个

约束的名字是 PERSON_GENDER_CK，类型为 CHECK，约束条件为 gender = 'F'；第 2 个约束的名字是 PERSON_AGE_CK，类型也为 CHECK，约束条件为 age BETWEEN 18 AND 35。其实在这个例子中只要看到了约束名，除了约束条件之外，您应该已经猜出了这两个约束的其他信息。

现在可以通过向 person 表中插入数据来测试您刚创建的条件（CHECK）约束。您可以使用例 13-73 的 DML 语句试着向 person 表中插入一行正常的数据（满足约束条件的数据）。

例 13-73

```
SQL> INSERT INTO person(id, name, gender, age)
   2  VALUES           (1001, '白小丫','F', 22);
```

例 13-73 结果

```
已创建 1 行。
```

例 13-73 的显示结果表明，您已经成功地把这行数据插入到 person 表中，这是因为白小丫是一位 20 出头的妙龄女郎，所以满足公司的招工要求（即满足约束条件）。

现在您可以使用例 13-74 的 DML 语句试着向 person 表中插入一行非正常的数据（不满足约束条件的数据）。

例 13-74

```
SQL> INSERT INTO person(id, name, gender, age)
   2  VALUES           (1002, '王老五', 'M', 19);
```

例 13-74 结果

```
INSERT INTO person(id, name, gender, age)
*
ERROR 位于第 1 行:
ORA-02290: 违反检查约束条件 (SCOTT.PERSON_GENDER_CK)
```

例 13-74 的显示结果表明，Oracle 系统拒绝执行您输入的 INSERT 语句。当您看完了 Oracle 系统显示的出错提示之后，重新审查了一下您的 INSERT 语句，终于恍然大悟。虽然王老五年龄合适，但他是一位男士，这当然不符合公司的招工要求（即不满足约束条件）。

现在您再使用例 13-75 的 INSERT 语句试着向 person 表中插入另一行非正常的数据（不满足约束条件的数据），看看 Oracle 系统会有什么反应。

例 13-75

```
SQL> INSERT INTO person(id, name, gender, age)
   2  VALUES           (1001, '李媛媛', 'F', 36);
```

例 13-75 结果

```
INSERT INTO person(id, name, gender, age)
*
ERROR 位于第 1 行:
ORA-02290: 违反检查约束条件 (SCOTT.PERSON_AGE_CK)
```

例 13-75 的显示结果表明，Oracle 系统又一次拒绝执行您输入的 INSERT 语句。虽然

李媛媛是一位相貌端庄的女士，但按公司的标准年龄超过 35 岁，这当然不符合公司的招工要求（即不满足约束条件）。

现在您再使用例 13-76 的 INSERT 语句试着向 person 表中插入另一行非正常的数据(不满足约束条件的数据），看看 Oracle 系统会有什么反应。

例 13-76

```
SQL> INSERT INTO person(id, name, gender, age)
  2  VALUES          (1001, '赵莺莺', 'F', 17);
```

例 13-76 结果

```
INSERT INTO person(id, name, gender, age)
*
ERROR 位于第 1 行:
ORA-02290: 违反检查约束条件 (SCOTT.PERSON_AGE_CK)
```

例 13-76 的显示结果表明，Oracle 系统又一次拒绝执行您输入的 INSERT 语句。虽然赵莺莺是一位相貌端庄的妙龄女郎，但是一位年龄不足 18 的未成年少女。雇用未成年少女是违法的，这当然不符合公司的招工要求（即不满足约束条件），因为公司不想因此而吃官司。

我们并未给出利用 UPDATE 语句来测试条件（CHECK）约束的例子，如果读者感兴趣可自己试着做一下。

与非空（NOT NULL）和唯一（UNIQUE KEY）约束一样，您不用测试 DELETE 语句，因为该语句不会产生数据违反条件（CHECK）约束的情形。

13.13　主 键 约 束

主键（PRIMARY KEY）是关系型数据库中一个非常重要的概念，您一旦在表中的某一列或某几列上定义了主键（PRIMARY KEY）约束，Oracle 系统就会自动地维护实体完整性。

还是老调重弹，我们还是用例子来解释主键（PRIMARY KEY）约束。在本节中我们还是使用 deptcon 表。为了做到心中有数，您应该先使用例 13-77 的查询语句来查看 deptcon 表中的内容。

例 13-77

```
SQL> select * from deptcon;
```

例 13-77 结果

```
   DEPTNO DNAME          LOC
---------- -------------- ------
       10 ACCOUNTING     EIJING
       20                牛街
       20                狼山镇
```

现在您可以试着使用例 13-78 的 DDL 语句在 deptcon 表的 deptno 上添加一个主键

（PRIMARY KEY）约束。

例 13-78

```
SQL> ALTER TABLE deptcon
  2  ADD CONSTRAINT deptcon_deptno_pk
  3       PRIMARY KEY (deptno);
```

例 13-78 结果

```
ADD CONSTRAINT deptcon_deptno_pk
             *
ERROR 位于第 2 行:
ORA-02437: 无法验证 (SCOTT.DEPTCON_DEPTNO_PK) - 违反主键
```

例 13-78 的显示输出告诉您，deptcon 表中可能有违反主键（PRIMARY KEY）约束的数据。这时您重新浏览例 13-77 的结果，发现在 deptcon 表中您错误地将坐落在狼山镇的第 30 号部门置成了 20 号，于是您马上使用例 13-79 的 DML 语句将该部门号重置成 30。

例 13-79

```
SQL> UPDATE deptcon
  2  SET deptno = 30
  3  WHERE loc = '狼山镇';
```

例 13-79 结果

```
已更新 1 行。
```

当看到例 13-79 的结果“已更新 1 行”之后，您就可以再使用与例 13-78 完全相同的例 13-80 的 DDL 语句在 deptcon 表的 deptno 列上重新添加一个主键（PRIMARY KEY）约束。

例 13-80

```
SQL> ALTER TABLE deptcon
  2  ADD CONSTRAINT deptcon_deptno_pk
  3       PRIMARY KEY (deptno);
```

例 13-80 结果

```
表已更改。
```

例 13-80 的显示结果表明，您已成功地在 deptcon 表的 deptno 列上定义了主键（PRIMARY KEY）约束。为了慎重起见，您可以使用例 13-81~例 13-83 的查询语句来查看在 deptcon 表上创建的所有约束信息。

例 13-81

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2        search_condition
  3  FROM user_constraints
  4  WHERE table_name = 'DEPTCON';
```

例 13-81 结果

```
OWNER    CONSTRAINT_NAME       C TABLE_NAME SEARCH_CONDITION
-------- -------------------- - ---------- --------------
SCOTT    DEPTCON_DEPTNO_PK     P DEPTCON
SCOTT    DEPTCON_DNAME_UK      U DEPTCON
```

例 13-82

```
SQL> COL COLUMN_NAME FOR A10
```

例 13-83

```
SQL> SELECT owner, constraint_name, table_name, column_name, position
  2  FROM user_cons_columns
  3  WHERE table_name = 'DEPTCON';
```

例 13-83 结果

```
OWNER    CONSTRAINT_NAME       TABLE_NAME COLUMN_NAM     POSITION
-------- -------------------- ---------- ---------- ----------
SCOTT    DEPTCON_DEPTNO_PK     DEPTCON    DEPTNO              1
SCOTT    DEPTCON_DNAME_UK      DEPTCON    DNAME               1
```

例 13-81 和例 13-83 的显示结果表明，在 SCOTT 用户的 DEPTCON 表上定义了两个约束，第 1 个约束的名字是 DEPTCON_DEPTNO_PK，类型为 PRIMARY KEY，它是定义在 DEPTNO 列上的；第 2 个约束的名字是 DEPTCON_DNAME_UK，类型为 UNIQUE，它是定义在 DNAME 列上的。

例 13-78 已经验证了主键（PRIMARY KEY）约束不允许表中有两行相同的数据，那么主键（PRIMARY KEY）约束允许插入空值（NULL）吗？我们还是用例子来说明，首先您使用例 13-84 的 DML 语句向 deptcon 表中插入一行部门号为空的数据。

例 13-84

```
SQL> INSERT INTO deptcon(deptno, dname, loc)
  2  VALUES             (NULL, '公关', '方园广场');
```

例 13-84 结果

```
INSERT INTO deptcon(deptno, dname, loc)
*
ERROR 位于第 1 行:
ORA-01400: 无法将 NULL 插入 ("SCOTT"."DEPTCON"."DEPTNO")
```

例 13-84 的显示结果表明，主键（PRIMARY KEY）约束不允许插入空值（NULL），这也是它和唯一（UNIQUE）约束的不同点。现在只要您将 deptno 改为某个有意义的值，之后再使用与例 13-84 相似如例 13-85 的 DML 语句就可以把这行数据插入到 deptcon 表中。

例 13-85

```
SQL> INSERT INTO deptcon(deptno, dname, loc)
  2  VALUES             (88, '公关', '方园广场');
```

例 13-85 结果

```
已创建 1 行。
```

现在您可以使用例 13-86 的查询语句来查看 deptcon 表中的内容。

例 13-86

```
SQL> SELECT *
  2  FROM deptcon;
```

例 13-86 结果

```
    DEPTNO DNAME            LOC
---------- --------------- -------
        10 ACCOUNTING      BEIJING
        88 公关            方园广场
        20                 牛街
        30                 狼山镇
```

在 13.2 节讲过当用户在一个表上建立主键（PRIMARY KEY）或唯一（UNIQUE）约束时，Oracle 系统会自动地创建唯一索引（UNIQUE INDEX），那么怎样才能找到这些索引呢？您可以使用例 13-87~例 13-92 的语句来得到它们的信息。

例 13-87

```
SQL> COL INDEX_TYPE FOR A10
```

例 13-88

```
SQL> COL INDEX_TYPE FOR A25
```

例 13-89

```
SQL> COL index_name for a20
```

例 13-90

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, TABLE_NAME, UNIQUENESS
  2  FROM user_indexes
  3  WHERE table_name = 'DEPTCON';
```

例 13-90 结果

```
INDEX_NAME           INDEX_TYPE                  TABLE_NAME UNIQUENES
-------------------- ------------------------- ---------- -----
DEPTCON_DEPTNO_PK    NORMAL                      DEPTCON    UNIQUE
DEPTCON_DNAME_UK     NORMAL                      DEPTCON    UNIQUE
```

例 13-91

```
SQL> col column_name for a15
```

例 13-92

```
SQL> SELECT index_name, table_name, column_name, column_position
  2  FROM user_ind_columns
  3  WHERE table_name = 'DEPTCON';
```

例 13-92 结果

```
INDEX_NAME           TABLE_NAME COLUMN_NAME     COLUMN_POSITION
-------------------- ---------- --------------- -------------
DEPTCON_DNAME_UK     DEPTCON    DNAME                       1
DEPTCON_DEPTNO_PK    DEPTCON    DEPTNO                      1
```

从例 13-90 和例 13-92 的结果可知，在表 DEPTCON 上有两个唯一索引（UNIQUE INDEX），它们的名字与相对应的约束名完全相同。

我们也未给出利用 UPDATE 语句来测试的例子，如果读者感兴趣可自己试着做一下。

与非空（NOT NULL）、唯一（UNIQUE KEY）和（CHECK）约束一样，您不用测试 DELETE 语句，因为该语句不会产生数据违反主键（PRIMARY KEY）约束条件的情形，即 DELETE 语句不会造成数据违反实体完整性（Entity Integrity）。

13.14　外键约束

由于外键（FOREIGN KEY）约束是用来维护从表（Child Table）和主表（Parent Table）之间的引用完整性（Referential Integrity）的，所以外键（FOREIGN KEY）约束要涉及的不止一个表。正是由于这个原因使得外键（FOREIGN KEY）约束要比其他几种约束更难理解。因此，本书要用较长的篇幅来讲解外键（FOREIGN KEY）约束。

下面我们用例子来解释外键（FOREIGN KEY）约束。为了减少工作量，您可以使用例 13-93 的 DDL 语句来创建一个名为 empcon 的表。下面将使用 empcon 表和 deptcon 表来完成该节中所有的例子。

例 13-93

```
SQL> CREATE TABLE empcon
  2  AS
  3  SELECT *
  4  FROM emp_dml;
```

例 13-93 结果

```
表已创建。
```

现在您可以试着使用例 13-94 的 DDL 语句在 empcon 表的 deptno 上添加一个外键（FOREIGN KEY）约束。

例 13-94

```
SQL> ALTER TABLE empcon
  2  ADD CONSTRAINT empcon_deptno_fk
  3       FOREIGN KEY(deptno) REFERENCES deptcon(deptno);
```

例 13-94 结果

```
ADD CONSTRAINT empcon_deptno_fk
            *
ERROR 位于第 2 行:
ORA-02298: 无法验证 (SCOTT.EMPCON_DEPTNO_FK) - 未找到父项关键字
```

运气不好，又出错了，但例 13-94 的显示输出告诉您，empcon 表中可能有违反引用完整性（Referential Integrity）的数据。这时您可以使用例 13-95 的查询语句来查看 empcon 表中所有的部门号（deptno）。

例 13-95

```
SQL> SELECT DISTINCT deptno
  2  FROM empcon;
```

例 13-95 结果

```
   DEPTNO
----------
       10
       20
       30
       88
```

之后，使用例 13-96 的查询语句来查看 deptcon 表中所有的部门号（deptno）。

例 13-96

```
SQL> SELECT *
  2  FROM deptcon;
```

例 13-96 结果

```
   DEPTNO DNAME          LOC
---------- --------------- --------
       10 ACCOUNTING      BEIJING
       20                 牛街
       30                 狼山镇
```

当看到了例 13-95 和例 13-96 的结果时您立刻明白了，因为 empcon 表中部门号(deptno)为 88 的数据行指向在 deptcon 表中根本不存在，即 empcon 表中部门号（deptno）为 88 的数据违反了引用完整性（Referential Integrity）。

如果您记不清什么是引用完整性（Referential Integrity）了，请复习一下第 7 章的有关内容。

知道了问题所在就好办了。为了解决这一问题，您可以使用例 13-97 的 DML 语句向 deptcon 表中插入一行部门号（deptno）为 88 的数据。

例 13-97

```
SQL> INSERT INTO deptcon
  2  VALUES      (88, '保卫', '狮子街');
```

例 13-97 结果

```
已创建 1 行。
```

当看到例 13-97 的结果“已创建 1 行”之后，您就可以再使用与例 13-94 完全相同的例 13-98 的 DDL 语句在 empcon 表的 deptno 上重新添加一个外键（FOREIGN KEY）约束。

例 13-98

```
SQL> ALTER TABLE empcon
  2  ADD CONSTRAINT empcon_deptno_fk
  3       FOREIGN KEY(deptno) REFERENCES deptcon(deptno);
```

例 13-98 结果

```
表已更改。
```

尽管例 13-98 的结果显示“表已更改”，但是出于慎重起见，还是应该再使用例 13-103 和例 13-105 的查询语句来查看在 empcon 表上创建的所有约束信息。

例 13-99

```
SQL>  COL owner FOR A8
```

例 13-100

```
SQL> COL CONSTRAINT_NAME FOR A20
```

例 13-101

```
SQL> COL TABLE_NAME FOR A10
```

例 13-102

```
SQL> COL SEARCH_CONDITION FOR A25
```

例 13-103

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2       r_constraint_name
  3  FROM user_constraints
  4  WHERE table_name = 'EMPCON';
```

例 13-103 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME
-------- -------------------- - ---------- ----------------
SCOTT    EMPCON_DEPTNO_FK     R EMPCON     DEPTCON_DEPTNO_PK
```

例 13-104

```
SQL> COL COLUMN_NAME FOR A10
```

例 13-105

```
SQL> SELECT owner, constraint_name, table_name, column_name, position
  2  FROM user_cons_columns
  3  WHERE table_name = 'EMPCON';
```

例 13-105 结果

```
OWNER    CONSTRAINT_NAME      TABLE_NAME COLUMN_NAM  POSITION
-------- -------------------- ---------- ---------- ----------
SCOTT    EMPCON_DEPTNO_FK     EMPCON     DEPTNO              1
```

例 13-103 和例 13-105 的显示结果表明，在 SCOTT 用户的 EMPCON 表上定义了一个类型为外键（FOREIGN KEY）约束，该约束的名字是 EMPCON_DEPTNO_FK，它是定义在 DEPTNO 列上并引用了名为 DEPTCON_DEPTNO_PK 的约束。从约束 DEPTCON_DEPTNO_PK 的名字可以猜出，此约束是定义在 DEPTCON 表的 DEPTNO 上的主键（PRIMARY KEY）约束。

在 EMPCON 表的 DEPTNO 列上定义完了外键（FOREIGN KEY）约束之后，您就可以用例子来测试它了。

13.15　外键约束对INSERT语句的影响

为了测试外键（FOREIGN KEY）约束对 INSERT 语句的影响，您可以使用例 13-106 的 INSERT 语句向 empcon 表中插入一行不违反引用完整性（Referential Integrity）的正常数据。

例 13-106

```
SQL> INSERT INTO empcon(empno, ename, mgr, hiredate, sal, comm, deptno)
  2  VALUES          (1010, '白小丫', 7839, SYSDATE, 5000, 1500, 10);
```

例 13-106 结果

```
已创建 1 行。
```

正如例 13-106 结果所显示的那样，您已成功地插入这行没有违反引用完整性（Referential Integrity）的正常数据。

现在您试着使用例 13-107 的 INSERT 语句向 empcon 表中插入一行违反引用完整性（Referential Integrity）的数据。

例 13-107

```
SQL> INSERT INTO empcon(empno, ename, mgr, hiredate, sal, comm, deptno)
  2  VALUES          (1010, '李媛媛', 7839, SYSDATE, 5000, 1500, 15);
```

例 13-107 结果

```
INSERT INTO empcon(empno, ename, mgr, hiredate, sal, comm, deptno)
*
ERROR 位于第 1 行:
ORA-02291: 违反完整约束条件 (SCOTT.EMPCON_DEPTNO_FK) - 未找到父项关键字
```

看到了例 13-107 显示的出错提示信息，您马上意识到了在 deptcon 表中并没有第 15 号部门，即在 deptcon 表中没有任何一行记录的 deptno 列的值为 15。因为李媛媛是总裁助理，所以她应该与总裁同属于第 10 号部门。于是您将她的部门号改为 10 后，使用例 13-108 的 INSERT 语句重新向 empcon 表中插入了这行不违反引用完整性（Referential Integrity）的正常数据。

例 13-108

```
SQL> INSERT INTO empcon(empno, ename, mgr, hiredate, sal, comm, deptno)
  2  VALUES          (1010, '李媛媛', 7839, SYSDATE, 5000, 1500, 10);
```

例 13-108 结果

```
已创建 1 行。
```

看到了例 13-108 显示的结果“已创建 1 行。”之后，您心里踏实多了。现在可以使用例 13-109 的查询语句来查看李媛媛这行数据是否正确地插入到了 empcon 表中。

例 13-109

```
SQL> SELECT empno, ename, mgr, hiredate, sal, comm, deptno
  2  FROM empcon
  3  WHERE sal >= 5000;
```

例 13-109 结果

```
EMPNO ENAME         MGR HIREDATE         SAL       COMM     DEPTNO
------ ---------- ----- ---------- ---------- --------- ----------
  7839 KING             17-11月-81      5000                    10
  1010 白小丫        7839 13-2月 -03      5000      1500          10
  1010 李媛媛        7839 13-2月 -03      5000      1500          10
```

看到例 13-109 的查询结果您可以放心了，因为李媛媛这行数据已经被正确地插入到 empcon 表中。现在您也许会问，如果向 deptcon 表中插入数据的话，会不会有违反引用完整性（Referential Integrity）的情形发生呢？想想看答案是什么？

答案是不会的。这里 deptcon 表称为父表或主表（PARENT TABLE）。

结论：

在进行插入操作时，只有操作是在子表或从表（CHILD TABLE）这一端时才会产生违反引用完整性（Referential Integrity）的问题，而操作是在父表或主表（PARENT TABLE）端时则不会产生。

13.16　外键约束对DELETE语句的影响

关于外键（FOREIGN KEY）约束对 DELETE 语句的影响是否也能得出类似于在 13.12 节中那样的结论呢？下面我们还是通过例子来说明。您可以试着使用例 13-110 的 DELETE 语句删除 deptcon 表中的一行数据。

例 13-110

```
SQL> DELETE FROM deptcon
  2  WHERE deptno = '88';
```

例 13-110 结果

```
DELETE FROM deptcon
*
ERROR 位于第 1 行:
ORA-02292: 违反完整约束条件 (SCOTT.EMPCON_DEPTNO_FK) - 已找到子记录日志
```

看到了例 13-110 显示的出错提示信息，您马上意识到在 empcon 表中可能还有员工属于第 88 号部门，即在 empcon 表中可能有些记录的 deptno 列的值为 88。很显然这违反引用完整性（Referential Integrity）。

如果您删除了 empcon 表中的一行数据，会不会可能有违反引用完整性（Referential Integrity）的情形发生呢？想想看答案是什么？

答案同样也是不会的。设想一下在现实中，如果某个未成年孩子的父母犯了不赦之罪，在现代的文明社会里，在法庭妥善安排这位无辜的孩子之前是不能处决他们的。因为这样做的结果会造成未成年的孩子流浪街头，但是反过来如果某个未成年孩子犯了罪，法庭就用不着考虑如何妥善安排他（她）的父母。

Oracle 系统也采取了同样的策略，即如果表与表之间建立了外键（FOREIGN KEY）约束的话，当对父表(Parent Table)进行 DELETE 操作时，Oracle 系统将保证不会在子表(Child Table)中产生任何“孤儿”，即子表中任何一行的外键永远不会指向一个在父表中不存在的数据行。

☠ **结论：**

在进行删除操作时，只有操作是在父表或主表（PARENT TABLE）这一端时才会产生违反引用完整性（Referential Integrity）的问题，而操作是在子表或从表（CHILD TABLE）端时不会产生。

13.17　外键约束对UPDATE语句的影响

之所以将外键（FOREIGN KEY）约束对 UPDATE 语句的影响放在最后讨论，是因为它是最复杂的。下面我们通过例子来说明。

假设您的公司已被另一个更有实力的大企业并购了，新的 CEO 是一位不信邪的人。为了向员工们展示他无所畏惧的个性，上任的第 1 天就让您把第 88 号部门改成第 44 号。受到他的感染，您也无所畏惧地使用了例 13-111 的 UPDATE 语句把 deptcon 表中部门号（deptno）为 88 的记录修改为 44。

例 13-111

```
SQL> UPDATE deptcon
  2  SET deptno = 44
  3  WHERE deptno = 88;
```

例 13-111 结果

```
UPDATE deptcon
*
ERROR 位于第 1 行:
ORA-02292: 违反完整约束条件 (SCOTT.EMPCON_DEPTNO_FK) - 已找到子记录日志
```

看到了例 13-111 的结果后，您赶紧使用例 13-112 的查询语句来查看 empcon 表，查看 88 号部门中是否还有员工。

例 13-112

```
SQL> SELECT ename, ename, job, sal, deptno
  2  FROM empcon
  3  WHERE deptno = 88;
```

例 13-112 结果

```
ENAME      ENAME      JOB              SAL       DEPTNO
---------  ---------  ---------  ---------  ---------
童铁蛋     童铁蛋     MANAGER          800           88
童铁蛋     童铁蛋     CTO              800           88
王老五     王老五     MANAGER          800           88
吴秀刚     吴秀刚     MANAGER          800           88
```

例 13-112 的结果表明，在 empcon 表中有 4 行部门号为 88 的记录，这正是造成例 13-111 的 UPDATE 语句违反引用完整性（Referential Integrity）的原因。

知道了原因之后，您改变了策略，试着先修改子表 empcon。于是您使用了例 13-113 的 UPDATE 语句将 empcon 表中所有第 88 号部门的员工的部门号（deptno）改为 44。

例 13-113

```
SQL> UPDATE empcon
  2  SET deptno = 44
  3  WHERE deptno = 88;
```

例 13-113 结果

```
UPDATE empcon
*
```

```
ERROR 位于第 1 行:
ORA-02291: 违反完整约束条件 (SCOTT.EMPCON_DEPTNO_FK) - 未找到父项关键字
```

运气实在是太差了，又出错了。例 13-113 的显示输出告诉您，对表 empcon 的修改又违反了引用完整性（Referential Integrity），因为在 deptcon 表中没有第 44 号部门。这个 UPDATE 语句使得 empcon 表中原来为 88 号部门的数据指向了一个在 deptcon 表中根本不存在的部门。

实际上我们到了进退两难的境地。那么您可能会问如何才能完成新 CEO 教给您的重托呢？请不用着急，就像电影中所说的，您可以采取“曲线救国”的方法。您可以先使用例 13-114 的 UPDATE 语句将 empcon 表中所有第 88 号部门改为空（NULL）。

例 13-114

```
SQL> UPDATE empcon
  2  SET deptno = NULL
  3  WHERE deptno = 88;
```

例 13-114 结果

```
已更新 4 行。
```

现在您就可以使用例 13-115 的 UPDATE 语句将 deptcon 表中所有第 88 号部门改为 44。

例 13-115

```
SQL> UPDATE deptcon
  2  SET deptno = 44
  3  WHERE deptno = 88;
```

例 13-115 结果

```
已更新 1 行。
```

最后，别忘了您应该使用例 13-116 的 UPDATE 语句将 empcon 表中所有为空（NULL）的部门号改为 44。

例 13-116

```
SQL> UPDATE empcon
  2  SET deptno = 44
  3  WHERE deptno IS NULL;
```

例 13-116 结果

```
已更新 4 行。
```

显示屏上显示出例 13-116 的结果“已更新 4 行”，但为了慎重起见，您还是使用了如例 13-117 和例 13-118 的查询语句来查看您是否真的把 deptcon 和 empcon 表中的第 88 号部门改成了第 44 号部门。

例 13-117

```
SQL> SELECT *
  2  FROM deptcon;
```

例 13-117 结果

```
    DEPTNO  DNAME           LOC
----------  --------------  -----
        10  ACCOUNTING      BEIJING
        44  保卫             狮子街
        20                  牛街
        30                  狼山镇
```

例 13-118

```
SQL> SELECT ename, ename, job, sal, deptno
  2  FROM empcon
  3  WHERE deptno = 44;
```

例 13-118 结果

```
ENAME       ENAME       JOB              SAL        DEPTNO
----------  ----------  ----------  ----------  ---------
童铁蛋       童铁蛋       MANAGER          800           44
童铁蛋       童铁蛋       CTO              800           44
王老五       王老五       MANAGER          800           44
吴秀刚       吴秀刚       MANAGER          800           44
```

例 13-117 和例 13-118 显示的结果再一次证明，在公司中您作为一名 Oracle 高手的地位是无人能挑战的。

结论：

在进行修改操作时，操作无论是在父表（PARENT TABLE）还是在子表（CHILD TABLE）端都可能会产生违反引用完整性（Referential Integrity）的问题。

13.18　外键约束对DDL语句的影响

外键（FOREIGN KEY）约束除了对 DML 语句有影响之外，对 DDL 语句也会产生影响。下面我们还是通过例子来解释。您可以试着用例 13-119 的 DDL 语句删除 deptcon 表。

例 13-119

```
SQL> DROP TABLE deptcon;
```

例 13-119 结果

```
DROP TABLE deptcon
           *
ERROR 位于第 1 行:
ORA-02449: 表中的唯一/主键被外部关键字引用
```

刚看到例 13-119 的显示结果时可能感到不解，因为您所发的 DROP TABLE 没有任何

语法错误，而且表也确实存在。当您仔细地阅读了系统显示的出错提示信息“表中的唯一/主键被外部关键字引用”后，您开始意识到该表中的 deptno 列已经被用作 empcon 表的外键（FOREIGN KEY）。Oracle 系统为了维护数据库中数据的引用完整性（Referential Integrity），不允许您在 empcon 表中有数据依赖于（指向）deptcon 表时删除 deptcon 表。

在现实中也是一样。例如，一个政府机构要进行精简，要撤销某个部门，一定要在先妥善地安排好该部门的工作人员之后才能撤销该部门。可以想象，如果该部门的所有工作人员在某一个早上来上班时突然发现他们所在的部门已经不见了，而他们的办公室已经被租给了其他公司，接下来可能会发生什么呢？

除了 DROP TABLE 语句，TRUNCATE TABLE 语句和删除列也可能产生类似的情形。您可以试着使用例 13-120 的 DDL 语句截断 deptcon 表，用例 13-121 的语句删除 deptcon 表中的 deptno 列。

例 13-120

```
SQL> TRUNCATE TABLE deptcon;
```

例 13-120 结果

```
TRUNCATE TABLE deptcon
               *
ERROR 位于第 1 行:
ORA-02266: 表中的唯一/主键被启用的外部关键字引用
```

例 13-121

```
SQL> ALTER TABLE deptcon
   2  DROP COLUMN deptno;
```

例 13-121 结果

```
DROP COLUMN deptno
            *
ERROR 位于第 2 行:
ORA-12992: 无法删除父项关键字列
```

当然，删除 deptcon 表中的 deptno 列操作在现实中是不应该发生的，因为它本身为 deptcon 表的主键。

结论：

在进行删除整个表时，只有删除的是父表或主表（PARENT TABLE）时才会产生违反引用完整性（Referential Integrity）的问题，而操作的是子表或从表（CHILD TABLE）时不会产生。

通过本节或以上各节对外键（FOREIGN KEY）约束的讨论，您应该明白，当两个表利用外键（FOREIGN KEY）建立了联系之后，Oracle 系统会自动地检查对这两个表的每一个 DML 和 DDL 操作，Oracle 系统不会执行任何违反引用完整性（Referential Integrity）的操作，也就是所有违反引用完整性（Referential Integrity）的数据都被挡在了 Oracle 数据库的

大门之外。这样做的好处确实不少，如数据库更稳定、更容易维护等。但有时也带来了一些操作上的麻烦，如您真的要删除主表（Parent Table）中的一行数据和所有在从表（Child Table）中依赖于它的数据行。Oracle 数据库管理系统设计者们早已高瞻远瞩地预见到了，并提供了 ON DELETE SET NULL 和 ON DELETE CASCADE 子句来解决这类麻烦。

13.19　外键的ON DELETE SET NULL和 ON DELETE CASCADE子句

ON DELETE SET NULL 子句的作用是，当主表（Parent Table）中的一行数据被删除时，Oracle 系统会自动地将所有从表（Child Table）中依赖于它的数据记录的外键（FOREIGN KEY）改为空（NULL）。真的像 Oracle 说的那样吗？做了以下的例子您自然就会明白了。

为了在 empcon 表中外键（FOREIGN KEY）约束上加入 ON DELETE SET NULL 子句，您应该先使用例 13-122 的 DDL 语句来删除 empcon 表中 deptno 列上现有的外键（FOREIGN KEY）约束。

例 13-122

```
SQL> ALTER TABLE empcon
  2  DROP CONSTRAINT empcon_deptno_fk;
```

例 13-122 结果

```
表已更改。
```

然后，应再使用例 13-123 的 DDL 语句在 empcon 表中重新定义一个包含了 ON DELETE SET NULL 子句的外键（FOREIGN KEY）约束。

例 13-123

```
SQL> ALTER TABLE empcon
  2  ADD CONSTRAINT empcon_deptno_fk
  3       FOREIGN KEY(deptno) REFERENCES deptcon (deptno)
  4       ON DELETE SET NULL;
```

例 13-123 结果

```
表已更改。
```

假设经过公安和执法部门的严打和治理整顿之后，治安明显改善，可以说国泰民安。在这样的大好形势下，您的公司决定取消保卫部。新的 CEO 不但不信邪，还是一位社会责任感极强和极有同情心的管理者（可能是“十大杰出中年”）。他不想把保卫部的这些人推向社会而变成低保对象，也更不愿见到这些人的家庭因他们失去工作而陷入经济困境。因此，他决定先将这些员工进行在职培训，然后再把他们分别放到不同的适合他们的部门工作（祝您在现实中真的能遇上这样大慈大悲的活菩萨）。

那么在 Oracle 数据库中如何实现他这崇高的决策呢？因为您早已为执行这一“救人于

水火”的决策做好了准备(您已在例 13-123 的 empcon 表中重新定义了一个包含 ON DELETE SET NULL 子句的外键约束)。首先您应该使用例 13-124 和例 13-125 的查询语句来查看在 deptcon 表中第 44 号部门和 empcon 表中属于第 44 号部门员工的信息。

例 13-124

```
SQL> SELECT *
  2  FROM deptcon
  3  ORDER BY deptno;
```

例 13-124 结果

DEPTNO	DNAME	LOC
10	ACCOUNTING	BEIJING
20		牛街
30		狼山镇
44	保卫	狮子街

例 13-125

```
SQL> SELECT empno, ename, job, sal, deptno
  2  FROM empcon
  3  WHERE deptno > 20
  4  ORDER BY deptno;
```

例 13-125 结果

EMPNO	ENAME	JOB	SAL	DEPTNO
7499	ALLEN	SALESMAN	1600	30
7521	WARD	SALESMAN	1250	30
7654	MARTIN	SALESMAN	1250	30
7698	BLAKE	MANAGER	2850	30
7844	TURNER	SALESMAN	1500	30
7900	JAMES	CLERK	950	30
7800	童铁蛋	MANAGER	800	44
7810	童铁蛋	CTO	800	44
1001	王老五	MANAGER	800	44
1002	吴秀刚	MANAGER	800	44

已选择 10 行。

检查完 deptcon 和 empcon 表之后，就可以利用例 13-126 的 DML 语句来履行您的崇高使命了。

例 13-126

```
SQL> DELETE FROM deptcon
  2  WHERE deptno = 44;
```

例 13-126 结果

```
已删除 1 行。
```

您可以使用例 13-127 的查询语句来检查是否真的删除了 deptcon 表中的第 44 号部门的数据。

例 13-127

```
SQL> SELECT *
  2  FROM deptcon;
```

例 13-127 结果

```
    DEPTNO DNAME          LOC
---------- -------------- -------
        10 ACCOUNTING     BEIJING
        20                牛街
        30                狼山镇
```

从例 13-127 结果可知，第 44 号部门保卫部已经被成功地删除。您现在就可以再使用与例 13-125 完全一样的例 13-128 查询语句来查看 Oracle 系统是如何处理 empcon 表中相关的信息的。

例 13-128

```
SQL> SELECT empno, ename, job, sal, deptno
  2  FROM empcon
  3  WHERE deptno > 20
  4  ORDER BY deptno;
```

例 13-128 结果

```
     EMPNO ENAME      JOB              SAL     DEPTNO
---------- ---------- --------- ---------- ------
      7499 ALLEN      SALESMAN        1600         30
      7521 WARD       SALESMAN        1250         30
      7654 MARTIN     SALESMAN        1250         30
      7698 BLAKE      MANAGER         2850         30
      7844 TURNER     SALESMAN        1500         30
      7900 JAMES      CLERK            950         30
已选择 6 行。
```

例 13-128 显示的结果可能使您多少有点感到意外，第 44 号部门中的员工随着该部门的取消似乎也不见了。难道 Oracle 这么伟大的设计也会出现这样不可思议的错误吗？Oracle 怎么会错呢？其实又是空值（NULL）在作怪。您现在可以在例 13-128 的查询语句中加入例 13-129 的一个条件 OR deptno IS NULL 之后，就可以清楚地看到 Oracle 系统是如何处理 empcon 表中相关的信息的。

例 13-129

```
SQL> SELECT empno, ename, sal, job, deptno
  2  FROM empcon
  3  WHERE deptno > 20
  4  OR    deptno IS NULL;
```

例 13-129 结果

```
     EMPNO ENAME             SAL JOB           DEPTNO
---------- ---------- ---------- --------- ---------
      7499 ALLEN            1600 SALESMAN         30
      7521 WARD             1250 SALESMAN         30
      7654 MARTIN           1250 SALESMAN         30
      7698 BLAKE            2850 MANAGER          30
      7844 TURNER           1500 SALESMAN         30
      7900 JAMES             950  CLERK            30
      7800 童铁蛋             800  MANAGER
      7810 童铁蛋             800  CTO
      1001 王老五             800  MANAGER
      1002 吴秀刚             800  MANAGER
已选择 10 行。
```

为了后面演示方便，我们先使用例 13-130 的语句来回滚刚做过的 DML 操作。

例 13-130

```
SQL> rollback;
```

例 13-130 结果

```
回退已完成。
```

然后，使用例 13-131 的 DDL 语句将原来定义在 empcon 表中的 deptno 列上的外键删除。

例 13-131

```
SQL> ALTER TABLE empcon
  2  DROP CONSTRAINT empcon_deptno_fk;
```

例 13-131 结果

```
表已更改。
```

现在您可以使用例 13-132 的 DDL 语句重新在 empcon 表中的 deptno 列上定义一个包含了 ON DELETE CASCADE 子句的外键。

例 13-132

```
SQL> ALTER TABLE empcon
  2  ADD CONSTRAINT empcon_deptno_fk
  3       FOREIGN KEY(deptno) REFERENCES deptcon (deptno)
  4       ON DELETE CASCADE;
```

例 13-132 结果

```
表已更改。
```

现在您可以再次利用例 13-133 的 DML 语句来删除同样的第 44 号部门。

例 13-133

```
SQL> DELETE FROM deptcon
  2  WHERE deptno = 44;
```

例 13-133 结果

```
已删除 1 行。
```

可以使用例 13-134 的查询语句来检查您是否真的删除了 deptcon 表中的第 44 号部门的数据。

例 13-134

```
SQL> SELECT *
  2  FROM deptcon;
```

例 13-134 结果

```
    DEPTNO DNAME                LOC
---------- --------------- -------
        10 ACCOUNTING           BEIJING
        20                      牛街
        30                      狼山镇
```

从例 13-134 结果可知，第 44 号的保卫部已经被成功地删除。您现在应该再使用例 13-135 的查询语句来查看这次 Oracle 系统是如何处理 empcon 表中相关的信息的。

例 13-135

```
SQL> SELECT empno, ename, sal, job, deptno
  2  FROM empcon;
```

例 13-135 结果

```
     EMPNO ENAME                SAL JOB              DEPTNO
---------- --------- ---------- -------- ---------
      7369 SMITH                800 CLERK                20
      7499 ALLEN                1600 SALESMAN            30
      7521 WARD                 1250 SALESMAN            30
      7566 JONES                2975 MANAGER             20
      7654 MARTIN               1250 SALESMAN            30
      7698 BLAKE                2850 MANAGER             30
      7782 CLARK                2450 MANAGER             10
      7788 SCOTT                3000 ANALYST             20
      7839 KING                 5000 PRESIDENT           10
```

```
      7844 TURNER              1500 SALESMAN         30
      7876 ADAMS               1100 CLERK            20
      7900 JAMES               950  CLERK            30
      7902 FORD                3000 ANALYST          20
      7934 MILLER              1300 CLERK            10
      1010 白小丫              5000                  10
      1010 李媛媛              5000                  10
已选择 16 行。
```

例 13-135 结果表明，Oracle 系统已经删除了 empcon 表中所有部门号为 44 的数据行。

在定义外键时，ON DELETE CASCADE 子句的使用要非常小心，它应该作为最后的选择。因为一旦您在定义外键时使用了 ON DELETE CASCADE 语句，任何在主表（Parent Table）上的误删除操作，哪怕只删除了一行，都可能造成在从表（Child Table）中成百上千，乃至上万行数据的丢失。

ON DELETE SET NULL 子句的副作用比 ON DELETE CASCADE 子句要小，但是也要谨慎使用，只有在不得已时才建议使用。

☠ **结论：**

如果在外键的定义中使用了 ON DELETE SET NULL 子句或 ON DELETE CASCADE 子句，无论删除操作是在父表（PARENT TABLE）这一端还是在子表或从表（CHILD TABLE）这一端都不会产生违反引用完整性（Referential Integrity）的问题，但是却留下了安全隐患。

13.20 约束的维护

在前面我们讲过为了维护数据的一致性，您可能会在一个表上定义一个或多个约束。但 Oracle 系统进行过多的约束检查会大大降低 Oracle 数据库系统的效率。在某些情况下，如在数据库系统中大规模装入数据时，为了系统的效率您不得不牺牲数据的一致性来关闭一些约束，甚至删除一些约束。

关闭约束的命令格式如下：

ALTER TABLE 表

DISABLE CONSTRAINT 约束名 [CASCADE];

其中，CASCADE 子句用来关闭存在有完整性关系的约束。DISABLE 子句既可以用在 CREATE TABLE 语句中，也可以用在 ALTER TABLE 语句中。

我们用以下的例子来演示如何关闭约束及如何得到该约束的相关信息。为了能使例子显示操作的复杂性，这里我们使用了对主表（Parent Table）中主键的操作。

首先应使用例 13-136 和例 13-137 的查询语句来查看您要操作的约束的相关信息（一定要包括状态 STATUS）。

例 13-136

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'EMPCON';
```

例 13-136 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- ------------------- - ---------- ------------------- -------
SCOTT    EMPCON_DEPTNO_FK     R EMPCON     DEPTCON_DEPTNO_PK    ENABLED
```

例 13-137

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'DEPTCON';
```

例 13-137 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- ----------------- - ---------- -------------------- -------
SCOTT    DEPTCON_DEPTNO_PK    P DEPTCON                        ENABLED
SCOTT    DEPTCON_DNAME_UK     U DEPTCON                        ENABLED
```

例 13-136 的结果表明，基于 empcon 表的外键 EMPCON_DEPTNO_FK 的状态（STATUS）为开启（ENABLED）。例 13-137 的结果表明，基于 deptcon 表的主键 DEPTCON_DEPTNO_PK 的状态（STATUS）也为开启（ENABLED）。

现在您可以试着使用例 13-138 的 DDL 语句来关闭主键约束 DEPTCON_DEPTNO_PK。

例 13-138

```
SQL> ALTER TABLE deptcon
  2  DISABLE CONSTRAINT deptcon_deptno_pk;
```

例 13-138 结果

```
ALTER TABLE deptcon
*
ERROR 位于第 1 行:
ORA-02297: 无法禁用约束条件 (SCOTT.DEPTCON_DEPTNO_PK) - 存在依赖关系
```

Oracle 系统拒绝执行您所发的 DDL 语句，但是从例 13-138 的结果给出的出错提示信息您可能已经猜到了，出错是因为在对应的从表（Child Table）中有外键指向主键 DEPTCON_DEPTNO_PK 所在的列。

这时 CASCADE 选项就派上了用场，您可以使用例 13-139 的含有 CASCADE 选项的 DDL 语句来关闭主键约束 DEPTCON_DEPTNO_PK。

例 13-139

```
SQL> ALTER TABLE deptcon
  2  DISABLE CONSTRAINT deptcon_deptno_pk CASCADE;
```

例 13-139 结果

```
表已更改。
```

作为一名优秀的 Oracle 数据库管理员或开发人员，您应该再次使用例 13-140 和例 13-141 的查询语句来查看您要操作的约束的相关信息（一定要包括状态 STATUS）。

例 13-140

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'DEPTCON';
```

例 13-140 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- ---------------- - ---------- -------------------- --------
SCOTT    DEPTCON_DEPTNO_PK    P DEPTCON                      DISABLED
SCOTT    DEPTCON_DNAME_UK     U DEPTCON                      ENABLED
```

例 13-141

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'EMPCON';
```

例 13-141 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- ------------------- - ---------- ----------------- --------
SCOTT    EMPCON_DEPTNO_FK     R EMPCON     DEPTCON_DEPTNO_PK    DISABLED
```

例 13-141 的结果表明，基于 empcon 表的外键 EMPCON_DEPTNO_FK 的状态（STATUS）已变为关闭(DISABLED)。例 13-140 的结果表明，基于 deptcon 表的主键 DEPTCON_DEPTNO_PK 的状态（STATUS）也已变为关闭（DISABLED）。

当您为了系统的效率而关闭了约束时，一些不一致的数据就有可能进到数据库中。如果数据库长时间地运行在这样的状态下，数据库可能会变得无法维护。因此在夜深人静（或数据库的联机操作很少）时，您应该重新打开您所关闭的约束。

打开约束的命令格式如下：

ALTER TABLE 表

ENABLE CONSTRAINT 约束名;

ENABLE 子句既可以用在 CREATE TABLE 语句中，也可以用在 ALTER TABLE 语句

中。如果您打开 UNIQUE KEY 或 PRIMARY KEY 约束，Oracle 系统将自动地为 UNIQUE KEY 或 PRIMARY KEY 建立索引（INDEX）。

当多数人在休息或已进入梦乡时，为了那温馨的小家庭您不得不继续工作。您现在可以使用例 13-142 的 DDL 语句重新打开（ENABLED）主键约束 DEPTCON_DEPTNO_PK，让 Oracle 系统来检测数据的一致性。

例 13-142

```
SQL> ALTER TABLE deptcon
  2  ENABLE CONSTRAINT deptcon_deptno_pk;
```

例 13-142 结果

```
表已更改。
```

作为一名优秀的 Oracle 数据库管理员或开发人员，您应再使用例 13-143 和例 13-144 的查询语句来查看您所操作过的约束的相关信息（一定要包括状态 STATUS）。

例 13-143

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,

  2        r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'DEPTCON';
```

例 13-143 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- ----------------- - --------- -------------------- -------
SCOTT    DEPTCON_DEPTNO_PK    P DEPTCON                         ENABLED
SCOTT    DEPTCON_DNAME_UK     U DEPTCON                         ENABLED
```

例 13-144

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2        r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'EMPCON';
```

例 13-144 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- ----------------- - --------- -------------------- --------
SCOTT    EMPCON_DEPTNO_FK     R EMPCON     DEPTCON_DEPTNO_PK    DISABLED
```

例 13-143 和例 13-144 结果表明，虽然基于 deptcon 表中 deptno 列的主键（PRIMARY KEY）已经开启（ENABLED），但是基于 empcon 表中 deptno 列的外键（FOREIN KEY）仍然为关闭（DISABLED）。

如果您开启（ENABLING）的主键（PRIMARY KEY）是用 CASCADE 选项关闭（DISABLED）的，Oracle 系统并不自动地开启（ENABLING）依赖于该主键（PRIMARY

KEY）的外键（FOREIN KEY）。您可以使用例 13-145 的 DDL 语句重新打开（ENABLING）外键（FOREIN KEY）约束 EMPCON_DEPTNO_FK。

例 13-145

```
SQL> ALTER TABLE empcon
  2  ENABLE CONSTRAINT empcon_deptno_fk;
```

例 13-145 结果

```
表已更改。
```

现在应该再次使用例 13-146 的查询语句来查看您刚操作过的外键（FOREIN KEY）约束 EMPCON_DEPTNO_FK 的相关信息（一定要包括状态 STATUS）。

例 13-146

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2        r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'EMPCON';
```

例 13-146 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- -------------------- - --------- -------------------- --------
SCOTT    EMPCON_DEPTNO_FK     R EMPCON    DEPTCON_DEPTNO_PK    ENABLED
```

例 13-146 结果表明，基于 empcon 表中 deptno 列的外键（FOREIN KEY）也已经开启（ENABLED）。

在 Oracle 数据库中，您不能修改约束。如果您想修改某一约束，需先将它删除，之后再重建。如果某一约束不再需要了，您也应该立即将其删除。

删除约束的命令格式如下：

ALTER TABLE 表

DROP CONSTRAINT 约束名 [CASCADE];

现在您可以试着使用例 13-147 的 DDL 语句将主键约束 DEPTCON_DEPTNO_PK 删除。

例 13-147

```
SQL> ALTER TABLE deptcon
  2  DROP  CONSTRAINT deptcon_deptno_pk;
```

例 13-147 结果

```
DROP  CONSTRAINT deptcon_deptno_pk
                *
ERROR 位于第 2 行:
ORA-02273: 此唯一/主键已被某些外部关键字引用
```

Oracle 系统又一次拒绝执行您所输入的 DDL 语句，但是从例 13-147 结果给出的出错提示信息您应该已经猜到了，有对应的从表（Child Table）中的外键指向主键 DEPTCON_

DEPTNO_PK 所在的列。

这时 CASCADE 选项又一次派上了用场，您可以使用例 13-148 的含有 CASCADE 选项的 DDL 语句来删除主键约束 DEPTCON_DEPTNO_PK。

例 13-148

```
SQL> ALTER TABLE deptcon
  2  DROP  CONSTRAINT deptcon_deptno_pk CASCADE;
```

例 13-148 结果

```
表已更改。
```

作为一名身经百战的 Oracle 专业人员，看到例 13-148 显示的结果“表已更改。”您还是不放心，于是又使用了例 13-149 和例 13-150 的查询语句来查看您所操作过的约束是否已经真的被删除。

例 13-149

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'DEPTCON';
```

例 13-149 结果

```
OWNER    CONSTRAINT_NAME      C TABLE_NAME R_CONSTRAINT_NAME    STATUS
-------- -------------------- - ---------- -------------------- --------
SCOTT    DEPTCON_DNAME_UK     U DEPTCON                         ENABLED
```

例 13-150

```
SQL> SELECT owner, constraint_name, constraint_type, table_name,
  2         r_constraint_name, status
  3  FROM user_constraints
  4  WHERE table_name = 'EMPCON';
```

例 13-150 结果

```
未选定行
```

例 13-149 和例 13-150 显示的结果再一次证明了您不愧是公司 Oracle 方面的泰斗。

13.21　约 束 小 结

您可能已经注意到了在例 13-109 结果中李媛媛和白小丫的 EMPNO 都为 1010，这是因为我们并没有在 EMPNO 上定义主键（PRIMARY KEY）约束。虽然作为教学的例子为了解释方便可以这样做，但这不是一个好的开发信息系统的方法。

我曾接触过一些开发商，有些开发人员为了避免由于不满足约束条件而不能进行某些 DML 或 DDL 操作的情况发生，他们几乎不在表上创建任何约束（包括主键和外键），而

当使用这些表时再用过程或函数等来检查所需的数据。这种设计有如下的弊端：

（1）不能在错误刚一出现时就发现。其后果在大型系统中可能是灾难性的，因为当一个表中的数据达到几十万行乃至几百万行之后再想找到那个错误简直成了大海捞针。

（2）无论是过程，还是函数，都是由程序员（开发人员）写的。他们的水平高过 Oracle 的设计者的可能性不大。因此，这些程序出错的可能性明显增加，执行效率也很难和 Oracle 提供的约束相比。

（3）过程或函数存储次数明显高于 Oracle 的约束，这无疑也会拖累系统的效率。

综上所述，您应该在创建表时就定义好所需的各种 Oracle 的约束。如果没有的话，应尽可能早地加上所需要的 Oracle 约束。这样会使系统更可靠，更容易维护。另外，如果在 Oracle 提供的 5 种约束和自定义的程序（触发器、过程或函数）两者之间有选择的话，应尽可能使用 Oracle 的约束，这样可能会改进系统的效率，增加系统的稳定性。

我们在前几节中使用了大量的例子来演示各种约束对不同的 DML 和 DDL 操作的影响。如果您觉得很难记的话，可以先记住上一段话和每一节的结论部分。之所以使用了那么大的篇幅来讲解，一是为了帮助读者理解各种不同的约束，特别是实体完整性（Entity Integrity）和引用完整性（Referential Integrity）这两个极其重要的概念；二是为了证明以上所有的结论不是“空谈”，而是“实话实说”。

细心的读者可能已经注意到了，与前面其他各章相比，这一章的篇幅长了许多。如果您读过其他类似的书，也会发现本书这一章的篇幅明显偏长。因为我在教学中和在 Oracle 的实际工作中发现不少学生甚至 Oracle 的从业人员对 Oracle 的约束的理解是相当肤浅的，这可能是因为 Oracle 的约束的概念看起来很简单，但要真正地理解并不容易，特别是当两个或更多个表建立起了主键（PRIMARY KEY）和外键（FOREIN KEY）的联系时。

Oracle 的约束，特别是主键（PRIMARY KEY）和外键（FOREIN KEY）约束及实体完整性（Entity Integrity）和引用完整性（Referential Integrity），这些是关系数据库中极其重要的概念，如果对这些概念没理解透的话，要设计和开发一个好的信息系统与“空谈”差不多，所以我希望读者能多花些时间把这些概念理解透彻。

13.22 应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 引入索引的原因。
- 如何手工建立索引？
- Oracle 系统如何自动地建立索引？
- 在系统中使用索引的利弊。
- 利用数据字典 user_indexes 和 user_ind_columns 获得索引信息。
- 如何维护索引？
- 引入约束的原因。
- 5 种约束的定义。

- 应该何时在表上定义约束？
- 主键与实体完整性。
- 外键与引用完整性。
- 在 INSERT 操作中 Oracle 系统是如何维护引用完整性的？
- 在 DELETE 操作中 Oracle 系统是如何维护引用完整性的？
- 在 UPDATE 操作中 Oracle 系统是如何维护引用完整性的？
- 在 DDL 操作中 Oracle 系统是如何维护引用完整性的？
- 如何使用外键的 ON DELETE SET NULL 和 ON DELETE CASCADE 子句？
- 如何利用 user_constraints 和 user_cons_columns 来得到约束的信息？
- 如何维护约束？

第14章

视图

什么是视图（Views）？视图是用户所看到的（数据）图像，这个图像并不是真正的数据，是经过转换后的一种数据表示。就像您在电视中看到的美女和英雄一样，观众看到的是经过许多化妆和包装后的人物，他/她们并不存在于现实生活中，观众看到的只是理想中的幻影（视图）。

14.1　为什么引入视图

如果您读过其他类似的书，可能会看到这些书在介绍视图时列举了许多引入视图的原因。其中最重要的原因是维护数据的独立性。那么什么是数据的独立性呢？

早期信息系统的设计与开发多采用模块驱动方式，即设计与开发者首先根据用户的需求建立功能模块（过程和函数），再由这些模块来决定数据结构和存储。以这种方法设计和开发的信息系统在绝大多数情况下是很难维护的。因为在当今这个经济快速发展的社会中，商业环境和市场在不断变化，用户的需求也在不断变化。当然带来的后果也就可想而知了，即相应的模块也要变，随之而来的是数据结构和数据存储都要变。可以说是牵一发而动全身。在现代化的社会中，变化是不可避免的，也许当今世界上唯一不变的是“变”这个字。

面对变化是不可避免的这一事实时，设计与开发者如何才能设计和开发出相对稳定的信息系统呢？即信息系统的某一局部变化时，不需要修改相应的数据结构和数据存储。研究表明：虽然随着用户的需求的变化，模块的变化几乎是不可避免的，但是信息系统中的数据却几乎是不变的（数据的排列可能会变化）。这样的发现就导致了一种与模块驱动方式完全不同的信息系统设计与开发方法的诞生，这种信息系统设计与开发方法就叫做数据驱动法。

采用数据驱动方式来设计和开发信息系统时，设计和开发者在前期可以完全不考虑系统的硬件、软件（包括操作系统和数据库管理系统）和功能模块（因此也就不用考虑程序设计语言）。设计和开发者在前期要收集信息系统所需的全部数据、划分实体（表）以及定义表与表之间的关系。然后用规范化（Normalization）决定哪些列应该放在哪个表中，最后产生一个基本上只包括了三范式表的实体关系图。当然这是一个漫长和需要多次反复的过程。

许多行家认为一旦完成了这一步，信息系统的设计已完成了一半或以上。因为接下去的工作已经很轻松了，只要将这个图中的实体转换成关系数据库的表，将实体与实体之间

的关系转换成关系数据库的约束即可。然后，只要将数据装入相应的表中，就可以利用 SQL 语句使用这一信息系统了，只是用户界面不太友好而已。

下面我们用一个简化的信息系统设计来说明采用数据驱动方式来设计和开发信息系统给信息系统维护带来的好处，如图 14.1 所示。

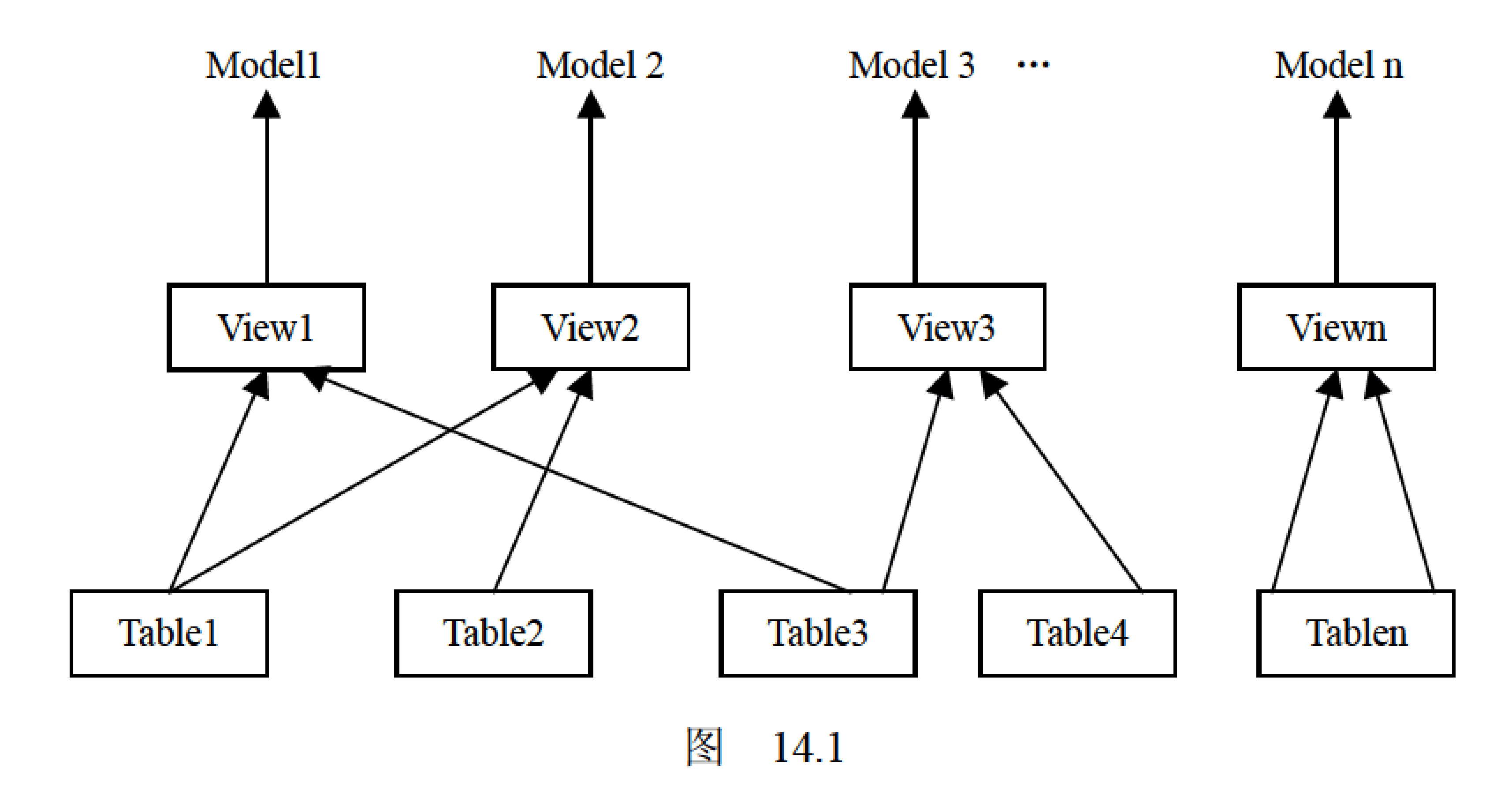


图　14.1

假设在图 14.1 中包含了信息系统所需的全部数据，而且每一个表都是三范式。很显然某一模块所需的数据可能分别存在于多个表中（如订单），这时就可以为这一模块创建一个视图，开发人员只能看到和操作通过这一视图所看到的数据，虽然这些数据是来自不同的表，但对于开发人员来说这些数据就好像来自一个表一样。当模块的功能变化需要不同的数据时，需要改变的只是相应的视图而不需要改变真正的表。这样，信息系统的维护就简单多了。

14.2　使用视图的好处

假设您的公司经营前景并不乐观，因此公司高级管理层需要进行严格的成本控制。老总要定期查看每一个部门员工的平均工资、平均佣金和员工的人数。于是老总先让您为他写个查询以得到这些信息。您试着写出了例 14-1 的查询语句。

例 14-1

```
SQL> SELECT d.dname, AVG(e.sal), AVG(NVL(comm,0)), COUNT(*)
  2  FROM emp e, dept d
  3  WHERE e.deptno = d.deptno
  4  GROUP BY d.dname;
```

例 14-1 结果

```
DNAME          AVG(E.SAL) AVG(NVL(COMM,0))   COUNT(*)
-------------- ---------- ---------------- ----------
ACCOUNTING     2916.66667                0          3
RESEARCH             2175                0          5
SALES          1566.66667       366.666667          6
```

毫无疑问，例 14-1 显示的结果就是您的老总所期望的，但是每一列的标题却令人费解。当然您可以通过为每一列设立一个别名来解决这一问题。

进一步假设，您的老总最近突然对 Oracle 数据库产生了浓厚的兴趣，他也想学习如何自己亲手写一些查询语句，他想从解决实际的问题开始，因此，他请您教他如何写您刚写的查询语句，而且他还想让每一列的标题都按他的习惯显示。

听了老总的话，您可能受宠若惊。不知苦苦地等待了多少个春秋，这一激动人心的时刻终于盼来了。高升之门终于在您面前打开了小小的一道缝。兴奋之余您也感到了一种巨大的压力，因为您的老总对 Oracle 数据库除了价格以外是一无所知。您可以想象教他写的第 1 条 SQL 语句竟是比例 14-1 的查询语句还要难，这样会产生什么样的后果。您当然不想拿这个千载难逢的机会去冒险。

经过仔细思考，您决定先为他创建一个如例 14-2 的视图。

例 14-2

```
SQL> CREATE VIEW average
  2  AS
  3  SELECT d.dname "部 门", AVG(e.sal) "平均工资",
  4        AVG(NVL(comm,0)) "平均佣金", COUNT(*) "员工数"
  5  FROM emp e, dept d
  6  WHERE e.deptno = d.deptno
  7  GROUP BY d.dname;
```

例 14-2 结果

```
视图已建立。
```

虽然例 14-2 节显示结果告诉您视图已建立，但作为一位资深的 Oracle 专家您当然非常谨慎。您应该使用例 14-3 的 SQL*Plus 命令来查看该视图的结构。

例 14-3

```
SQL> DESC average
```

例 14-3 结果

```
名称                                        是否为空? 类型
----------------------------------------- -------- -----
部 门                                                VARCHAR2(14)
平均工资                                             NUMBER
平均佣金                                             NUMBER
员工数                                               NUMBER
```

例 14-3 显示结果表明，您所建立的视图正确无误。现在您就可以进行第 2 步，教您的老总写他一生中第 1 条 SQL 语句，如例 14-4。

例 14-4

```
SQL> SELECT *
  2  FROM average;
```

例 14-4 结果

部 门	平均工资	平均佣金	员工数
ACCOUNTING	2916.66667	0	3
RESEARCH	2175	0	5
SALES	1566.66667	366.666667	6

当您的老总看到他亲手输入的 SQL 语句获得了他所需要的结果时兴奋不已。他拍拍您的肩膀说："真没想到，在我的身边居然有这么好的 Oracle 老师。"他告诉您他参加了无数次的 Oracle 讲座，就没有一个人能像您这样用这么短的时间教会他用 Oracle 获得了他所需要的信息。听了老总的赞扬，您知道这回马屁是拍到点上了。您仿佛看到了自己已经踏上了成功之路。

从前面的讨论可以看出使用视图有以下的好处：

- 可以把复杂的 SQL 语句简单化。您可以看出例 14-2 中的 SQL 语句是很复杂的。在该语句中包含了分组函数（Group Functions），还包含了 GROUP BY 子句和两个表的连接，并为查询中的每一列取了别名。但用户却可使用像例 14-4 那样最简单的查询语句从视图 average 中提取同样的信息。
- 可以限制数据库的访问。用户通过视图 average 只能访问"部门"、"平均工资"、"平均佣金"和"员工数"这 4 列。
- 可以使数据独立于应用程序（数据独立性），见 14.1 节的讨论。
- 可以使相同的数据以不同的形式出现在不同的视图中，例如，您可以为每一个部门创建一个只包含该部门员工信息的视图，这样当用户使用视图来访问数据时就只能看到他/她所在部门员工的信息。

14.3　如何创建视图

实际上您已经在 14.2 节中创建了一个视图，下面我们给出创建视图比较完整的格式：

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW 视图名
　　[(别名[, 别名]…)]
AS
子查询语句
[WITH CHECK OPTION [CONSTRAINT 约束名]]
[WITH READ ONLY]

其中每个参数的含义如下。

- OR REPLACE：如果所创建的视图已存在，Oracle 系统会重建这个视图。在这里值得注意的一个问题是，可能系统中已有一个同名的视图，而且它是一个有用的视图，此时，您如果使用 OR REPLACE 来创建视图就会把系统中有用的视图覆盖掉。因此，为了防止这种错误的发生，您应该在创建视图之前使用 SQL*Plus 的

DESC 命令查看这个视图是否已存在。

- FORCE：不管所引用的表是否存在，Oracle 系统都会创建这个视图。
- NOFORCE：只有当所引用的表都存在时，Oracle 系统才会创建这个视图（这是 Oracle 系统的默认方式）。
- 视图名：所要创建的视图的名字。
- 别名：为视图所产生的列定义的列名（别名的个数一定要与视图所产生的列数相等）。
- 子查询语句：一个完整的查询语句（您也可以在该语句定义别名）。
- WITH CHECK OPTION：所插入或修改的数据行必须满足视图所定义的约束条件。
- 约束名：WITH CHECK OPTION 中的约束名。
- WITH READ ONLY：保证在该视图上不能进行任何 DML 操作。

需要注意的是，在子查询语句中不能包含 ORDER BY 子句，这可能是因为视图给出的数据并不是最终的结果。如果您想让您的查询结果按某一种有序的方式显示，您可以在查询视图时加入 ORDER BY 子句。

下面再通过一个例子来演示如何使用 CREATE OR REPLACE VIEW 语句来创建视图。假设您的老总在您的影响下越发觉得管理者了解和使用 Oracle 系统的重要性。为此他召开了一个由公司所有中层或以上经理（管理人员）参加的重要会议，您也被邀请列席。您的老总首先介绍了他学习和使用 Oracle 系统的宝贵经验。所有与会人员都对他如此惊人的学习速度羡慕不已（肯定也不乏阿谀奉承之人）。在众人的一片赞扬声中，他郑重地宣布要将他的成功经验在公司管理层进行推广。

幸运之神终于降临了，您终于有机会接近公司的权力核心，当天晚上您几乎是彻夜未眠。经过了一夜的周密思考，您决定您的下一个“学生”是公司的财务总监，因为他是仅次于老总的第 2 号人物，而且是老总的铁哥们。您首先使用例 14-5 的 SQL 语句为财会部门创建了一个名为 acct 的视图，其中包含了“名字”、“工资”、“职位”和“雇用日期”4 列。

例 14-5

```
SQL> CREATE OR REPLACE VIEW acct
  2  ("名 字", "工 资", "职 位", "雇用日期")
  3  AS
  4  SELECT ename, sal, job, hiredate
  5  FROM emp
  6  WHERE deptno = 10;
```

例 14-5 结果

```
视图已建立。
```

从现在起，每一步都可能事关您的锦绣前程，当然应该非常谨慎，所以您立刻使用了例 14-6 的 SQL*Plus 命令来查看刚建立的视图的结构，以确保在教财务总监时万无一失。

例 14-6

```
SQL> DESC acct
```

例 14-6 结果

```
名称                                    是否为空? 类型
-------------------------------------- --------- ------------
名 字                                              VARCHAR2(10)
工 资                                              NUMBER(7,2)
职 位                                              VARCHAR2(9)
雇用日期                                            DATE
```

当看到了例 14-6 显示的结果时您终于可以放心了，然后您开始教财务总监例 14-7 的第 1 条查询语句。

例 14-7

```
SQL> SELECT *
  2  FROM acct;
```

例 14-7 结果

```
名字                工资  职位            雇用日期
---------- ---------- ---------- -----------
CLARK            2450  MANAGER         09-6 月 -81
KING             5000  PRESIDENT       17-11 月-81
MILLER           1300  CLERK           23-1 月 -82
```

当财务总监亲手输入了例 14-7 的查询语句时，他欣喜若狂，因为这是他平生第 1 次自己从数据库中获得了他想要的如此重要的信息。您当然也是对他的聪明才智赞不绝口（心里没准想：这么简单的事，若是拴个饽饽狗都能干。但是为了自己的锦绣前程还是不得不说些违心话，不得已说假话上帝是会原谅的）。当然这只是万里长征迈出了可喜的一步。接下去您还要为其他的部门创建类似的视图。

您可能已经注意到了，在前面所有的例子中我们都为视图中的列取了别名，其原因有以下两条：

（1）一般表是 IT 专业人员使用的，为了减少输入的次数，表的设计者们一般倾向于用缩写形式来表示表名和列名。但是视图就不一样了，一般视图是非 IT 专业人员使用的，这时如果再使用缩写形式会令人望而生畏。因此，视图中的列一般要使用用户熟知的名字。有人说："在公司中关键的不在于您干得如何，而在于您的老板觉得您干得如何。"所以，面上的工作还是要用心去做的，有时可能会达到事半功倍的效果。

（2）一般表名和列名都是用 ASCII 码（即美国英文）定义的，而在视图中就应该为列定义用户熟知的别名（一般为用户的本国语言），这样可以增加易读性。

14.4 如何修改视图

当您的财务总监学会了如何使用 SQL 语句来获取他想要的信息之后，兴奋地将此事告

诉了您的老总，老总对他使用的视图也很感兴趣，但他希望通过这个视图还能查到部门的名字和地点，于是，他把这一想法告诉了您。看来您是鸿运当头了，公司的权力核心离您越来越近，您当然毫不犹豫地答应下来。您使用了例 14-8 的 DDL 语句完成了老总的重托。

例 14-8

```
SQL> CREATE OR REPLACE VIEW acct
  2  ("名 字", "工 资", "职 位", "雇用日期", "部 门", "地 点")
  3  AS
  4  SELECT ename, sal, job, hiredate, dname, loc
  5  FROM emp, dept
  6  WHERE emp.deptno = dept.deptno
  7  AND   emp.deptno = 10;
```

例 14-8 结果

```
视图已建立。
```

注意：

Oracle 并没有提供直接修改视图的方法。如果您想修改一个已存在的视图，您就要像例 14-8 那样重新输入 SQL 语句将原来的视图覆盖掉。

为了谨慎起见，您应该使用例 14-9 的 SQL*Plus 命令来查看刚建立的视图的结构，以确保在交给老总和财务总监时万无一失。

例 14-9

```
SQL> DESC acct
```

例 14-9 结果

```
名称                                  是否为空? 类型
----------------------------------- -------- ------------
名 字                                         VARCHAR2(10)
工 资                                         NUMBER(7,2)
职 位                                         VARCHAR2(9)
雇用日期                                      DATE
部 门                                         VARCHAR2(14)
地 点                                         VARCHAR2(13)
```

当看到了例 14-9 显示的结果时您终于可以放心了，然后，您告诉您的老总和财务总监可以使用新的 acct 视图了。随后财务总监当着老总的面执行了例 14-10 的查询语句。

例 14-10

```
SQL> SELECT *
  2  FROM acct;
```

例 14-10 结果

```
名 字                  工 资 职 位        雇用日期       部 门         地 点
---------- ---------- --------- ---------- -------------- --------
```

```
CLARK                      2450 MANAGER      09-6 月 -81  ACCOUNTING  NEW YORK
KING                       5000 PRESIDENT    17-11 月-81  ACCOUNTING  NEW YORK
MILLER                     1300 CLERK        23-1 月 -82  ACCOUNTING  NEW YORK
```

看到了例 14-10 显示的结果，您的老总和财务总监脸上都不约而同地露出了灿烂的笑容，因为他们终于发现他们所需要的信息就在他们的手中。但这种灿烂的笑容对您来说可能意义更加深远。

14.5 Oracle系统如何管理视图

当您使用 CREATE VIEW 语句成功地创建了视图后，这个视图就被存在了 Oracle 数据字典中。那么一个用户怎样才能知道他/她的账号下有多少个视图以及这些视图的定义呢？您可以通过查询数据字典 user_views 来得到这方面的信息（例 14-11）。

例 14-11

```
SQL> SELECT view_name, text_length, text
  2  FROM user_views;
```

例 14-11 结果

```
VIEW_NAME                      TEXT_LENGTH
------------------------------ ---------------
TEXT
-----------------------------------------------
ACCT                                    112
SELECT ename, sal, job, hiredate, dname, loc
FROM emp, dept
WHERE emp.deptno = d
AVERAGE                              162
SELECT d.dname "部 门", AVG(e.sal) "平均工资",
      AVG(NVL(comm,0)) "平均佣金
```

从例 14-11 显示的结果可以看到您刚创建的两个视图和它们的子查询语句。

从以上的例子可以看出，一旦您创建了一个视图之后，您就可以像使用表一样对它进行查询。其实，从实用的角度来看，表和视图几乎没什么区别，那么当您使用视图来进行查询时，Oracle 系统又是如何工作的呢？

Oracle 系统的执行步骤如下：

（1）从数据字典中取出视图的定义，即查询语句。

（2）检查该视图所引用的表的权限。

（3）执行视图所定义的查询语句。

从以上 Oracle 系统的执行步骤可以看出，在使用视图访问数据库时，至少要两次访问硬盘（第 1 次访问数据字典，第 2 次访问表中的数据）。因此，虽然使用视图给我们带来

了诸多的方便，但是它却可能带来一些效率方面的问题，因为磁盘的 I/O 操作对系统效率的冲击是非常大的。

14.6　如何使用视图来进行DML操作

在讨论如何使用视图（Views）来进行 DML 操作之前，我们先介绍视图（Views）的分类。视图（Views）分两类，即简单视图和复杂视图，它们的定义如下。

简单视图：

- 数据是仅从一个表中提取。
- 不包含函数。
- 不包含分组数据。
- 可以通过该视图进行 DML 操作。

复杂视图：

- 数据是从多个表中提取。
- 包含函数。
- 包含分组数据。
- 不一定能够通过该视图进行 DML 操作。

虽然您可以通过视图进行 DML 操作，但是 Oracle 系统加上了许多限制。实际上在视图上进行的 DML 操作都要转换成对所引用表的 DML 操作。所有通过视图进行 DML 操作的规则如下：

- 可以在简单视图上执行 DML 操作。
- 如果在一个视图中包含了分组函数、GROUP BY 子句或 DISTINCT 关键字，就不能通过该视图进行删除（DELETE）操作。
- 如果在一个视图中包含了分组函数、GROUP BY 子句或 DISTINCT 关键字，也不能通过该视图进行修改（UPDATE）操作。
- 如果在一个视图中包含了由表达式组成的列或伪列 ROWNUM，也不能通过该视图进行修改（UPDATE）操作。
- 如果在一个视图中包含了分组函数、GROUP BY 子句或 DISTINCT 关键字，不能通过该视图进行插入（INSERT）操作。
- 如果在一个视图中包含了由表达式组成的列或伪列 ROWNUM，也不能通过该视图进行插入（INSERT）操作。
- 如果在一个视图中没有包含引用表中那些不能为空（NOT NULL）的列，也不能通过该视图进行插入（INSERT）操作。

如果记住在视图上进行 DML 操作都要转换成对所引用表的 DML 操作和表中的列是不可分割的最小单位，就不难理解 Oracle 为什么要对通过视图进行 DML 操作进行如此之多的限制。

14.7　如何使用视图的WITH CHECK OPTION子句

为了演示 WITH CHECK OPTION 子句的用法，可以使用例 14-12 的 DDL 语句来创建一个含有 WHERE 子句和 WITH CHECK OPTION CONSTRAINT 子句的视图。

例 14-12

```
SQL> CREATE VIEW sales30
  2  AS
  3  SELECT *
  4  FROM emp
  5  WHERE deptno = 30
  6  WITH CHECK OPTION CONSTRAINT sales30_ck;
```

例 14-12 结果

```
视图已建立。
```

为了谨慎起见，您还是应该使用例 14-13 的 SQL*Plus 命令来查看刚建立的视图的结构。

例 14-13

```
SQL> DESC sales30
```

例 14-13 结果

```
名称                                       是否为空? 类型
----------------------------------------- -------- ------------
EMPNO                                      NOT NULL NUMBER(4)
ENAME                                               VARCHAR2(10)
JOB                                                 VARCHAR2(9)
MGR                                                 NUMBER(4)
HIREDATE                                            DATE
SAL                                                 NUMBER(7,2)
COMM                                                NUMBER(7,2)
DEPTNO                                              NUMBER(2)
```

例 14-13 显示的结果表明，您所创建的视图的结构完全正确，现在您还应该检查数据是否正确，如例 14-14 所示。

例 14-14

```
SQL> SELECT empno, ename, job, sal, deptno, comm, deptno
  2  FROM sales30;
```

例 14-14 结果

```
     EMPNO ENAME      JOB              SAL     DEPTNO      COMM     DEPTNO
---------- ---------- -------- ---------- ---------- -------- ------
```

```
      7499 ALLEN      SALESMAN        1600         30        300         30
      7521 WARD       SALESMAN        1250         30        500         30
      7654 MARTIN     SALESMAN        1250         30       1400         30
      7698 BLAKE      MANAGER         2850         30                    30
      7844 TURNER     SALESMAN        1500         30          0         30
      7900 JAMES      CLERK            950         30                    30
已选择 6 行。
```

当看到了例 14-14 显示的结果时您终于可以放心了。

设想一下您的老总为了管理上的方便（即开会时找人方便）要将所有经理的部门号（deptno）改为 10。当然这一光荣的任务又得由您来完成，您试着使用例 14-15 的 DML 语句来完成老总的重托。

例 14-15

```
SQL> UPDATE sales30
  2  SET deptno = 10
  3  WHERE job = 'MANAGER';
```

例 14-15 结果

```
UPDATE sales30
       *
ERROR 位于第 1 行:
ORA-01402: 视图 WITH CHECK OPTIDN 违反 where 子句
```

例 14-15 显示的结果告诉我们，您在例 14-15 所做的修改违反了您在创建视图时用 WHERE 子句所限定的条件（WHERE deptno = 30），所以 Oracle 系统拒绝执行您修改的语句。

视图中的 WITH CHECK OPTION CONSTRAINT 子句的含义是：所有通过该视图进行的 DML 操作都不能违反在创建视图时用 WHERE 子句所限定的条件。

还记得在第 12 章的 12.8 节中所谈的例子吗？人事经理不小心将一些新员工的工资压到了公司所允许的最低工资标准之下，结果是好心办了坏事（也可能在员工们看来本来就是狼心）。现在您可以使用带有 WITH CHECK OPTION CONSTRAINT 子句的视图来防止这种尴尬的局面发生，当然也是为了避免让公司特别是老板陷入不仁不义的境地。于是您使用例 14-16 的 DDL 语句创建了一个新的视图。

例 14-16

```
SQL> CREATE VIEW salary_limit
  2  AS
  3  SELECT *
  4  FROM emp
  5  WHERE sal >= 800
  6  WITH CHECK OPTION CONSTRAINT salary_limit_ck;
```

例 14-16 结果

```
视图已建立。
```

现在您就可以要求有关人员通过视图 salary_limit 来输入新员工的数据，这样就能防止违法行为的发生。

例 14-17

```
SQL> INSERT INTO salary_limit (empno, ename, job, mgr,
   2                  hiredate, sal, comm, deptno)
   3  VALUES             (7800, '童铁蛋', '保安',7900, SYSDATE, 666, 77, 66);
```

例 14-17 结果

```
INSERT INTO salary_limit (empno, ename, job, mgr,
          *
ERROR 位于第 1 行:
ORA-01402: 视图 WITH CHECK OPTIDN 违反 where 子句
```

从例 14-17 显示的结果您已经发现了，Oracle 系统不允许插入这行数据，因为它违反了在视图 salary_limit 中 WHERE 子句所规定的条件 sal >= 800（童铁蛋的工资只为 666）。

14.8　为什么要使用WITH READ ONLY子句

假设财务总监经过了一段时间的学习已经掌握了不少的 SQL 语句，其中也包括了 DML 语句，现在他想试试他的身手。为了安全起见他决定使用视图 acct 来做练习，于是他输入了例 14-18 的 UPDATE 语句。

例 14-18

```
SQL> UPDATE acct
   2  SET "工 资" = 9999;
```

例 14-18 结果

```
已更新 3 行。
```

当看到了例 14-18 显示的结果之后，他迫不及待地使用了例 14-19 的查询语句来检查他刚做过的修改。

例 14-19

```
SQL> SELECT *
   2  FROM acct;
```

例 14-19 结果

```
名 字                工 资 职 位       雇用日期      部 门          地 点
---------- ---------- ---------- ----------- -------------- ------
CLARK              9999 MANAGER    09-6 月 -81  ACCOUNTING    NEW YORK
KING               9999 PRESIDENT 17-11 月-81  ACCOUNTING    NEW YORK
MILLER             9999 CLERK      23-1 月 -82  ACCOUNTING    NEW YORK
```

当看到了例 14-19 显示的结果之后，他兴奋之极，因为他发现他已经能够成功地修改

数据库中的数据了。就在他兴高采烈地将这一切告诉您时，作为一位久经沙场的 Oracle DBA 您已经意识到问题的严重性，随即使用例 14-20 的查询语句来检查在 emp 表中有多少行的数据被修改了。

例 14-20

```
SQL> SELECT ename, sal, job
  2  FROM emp
  3  WHERE deptno = 10;
```

例 14-20 结果

```
ENAME             SAL JOB
---------- ---------- -------
CLARK            9999 MANAGER
KING             9999 PRESIDENT
MILLER           9999 CLERK
```

看到例 14-20 显示的结果之后，您立刻使用例 14-21 的 SQL 语句来回滚您的财务总监刚做过的修改。

例 14-21

```
SQL> ROLLBACK;
```

例 14-21 结果

```
回退已完成。
```

以上的例子表明，对一个视图中数据的修改都要转换成对视图所引用表的修改。其实，任何对视图中数据所进行的 DML 操作都要转换成对该视图所引用表的 DML 操作，因为视图中是没有存放任何数据的。

那么如何才能避免以上的错误再发生呢？答案是在创建视图时使用 WITH READ ONLY 子句。为了避免以上的错误发生，您利用如下包含了 WITH READ ONLY 子句的 CREATE VIEW 语句重建了例 14-22 的 acct 视图。

例 14-22

```
SQL> CREATE OR REPLACE VIEW acct
  2  ("名 字", "工 资", "职 位", "雇用日期", "部 门", "地 点")
  3  AS
  4  SELECT ename, sal, job, hiredate, dname, loc
  5  FROM emp, dept
  6  WHERE emp.deptno = dept.deptno
  7  AND   emp.deptno = 10
  8  WITH READ ONLY;
```

例 14-22 结果

```
视图已建立。
```

当视图 acct 被创建之后，您再试着使用例 14-23 的 UPDATE 语句修改 acct 视图中的数据。

例 14-23

```
SQL> UPDATE acct
   2  SET "工 资" = 9999;
```

例 14-23 结果

```
SET "工 资" = 9999
    *
ERROR 位于第 2 行:
ORA-01733: 此处不允许虚拟列
```

Oracle 拒绝执行您的 UPDATE 语句，因为您在创建视图 acct 时使用了 WITH READ ONLY 子句。这时如果您试图使用例 14-24 的 DELETE 语句来删除视图 acct 中的数据，Oracle 同样会拒绝执行您的 DELETE 语句。

例 14-24

```
SQL> DELETE FROM acct;
```

例 14-24 结果

```
DELETE FROM acct
            *
ERROR 位于第 1 行:
ORA-01752: 不能从没有一个键值保存表的视图中删除
```

其实，如果您在创建一个视图时使用了 WITH READ ONLY 子句，通过该视图所做的任何 DML 语句都将引起 Oracle 服务器产生错误。

建议：

在创建视图时应该尽可能地使用 WITH READ ONLY 子句，这样可以避免许多因 DML 误操作对真正表的破坏。只有当您确信必须使用视图来进行 DML 操作时，才不使用 WITH READ ONLY 子句。

14.9　如何删除视图

当一个视图没有用时，您可以使用 DROP VIEW 语句将该视图删除。删除视图并不像删除表那样危险，因为视图中没有数据，真正的数据是存在于所引用的表中。删除视图命令的格式如下：

DROP VIEW　视图名

为了进一步理解这一语句的含义，您可以使用例 14-25 的查询语句来检查您到底有多少视图。

例 14-25

```
SQL> SELECT view_name, text_length
   2  FROM user_views;
```

例 14-25 结果

```
VIEW_NAME                     TEXT_LENGTH
----------------------------  -----------
ACCT                                  127
AVERAGE                               162
SALARY_LIMIT                          112
SALES30                               113
```

例 14-25 的结果显示了您所拥有的每一个视图以及定义这个视图的语句中所包含的字符个数。假设您的财务总监已经不再需要视图 acct 了，此时您可以帮助他从 Oracle 系统中删除这一视图，因此，您使用了例 14-26 的 DDL 语句。

例 14-26

```
SQL> DROP VIEW acct;
```

例 14-26 结果

```
视图已丢掉。
```

现在您就可以如例 14-27 所示再次查询数据字典 user_views，查看视图 acct 是否真的被删除了。

例 14-27

```
SQL> SELECT view_name, text_length
  2  FROM user_views;
```

例 14-27 结果

```
VIEW_NAME                      TEXT_LENGTH
------------------------------ ---------
AVERAGE                               162
SALARY_LIMIT                          112
SALES30                               113
```

您也可以使用例 14-28 所示的 SQL*Plus 的命令 DESC 达到同样的目的。

例 14-28

```
SQL> DESC acct
```

例 14-28 结果

```
ERROR:
ORA-04043: 对象 acct 不存在
```

到此为止，我们已经介绍了有关视图（Views）的所有基本操作。下面将介绍一些高级功能，这些功能主要是为了支持决策支持系统（DSS）或数据仓库系统（Data Warehouse）而设计的。早期的 Oracle 数据库管理系统主要是为了支持联机事务处理系统（OLTP）而设计的。

虽然有关决策支持系统（DSS）和数据仓库系统（Data Warehouse）的理论在很早以前就已经成熟，但因为这种系统要消耗大量的硬件资源，而当时的硬件又很昂贵，所以几乎

没有多少企业有能力使用这种系统。到了 20 世记 90 年代，特别是 90 年代中期，情况发生了巨大的变化，计算机硬件的价格持续暴跌但性能却稳定上升，这就为决策支持系统和数据仓库系统的实际应用铺平了道路。

正是在这种形式下，Oracle 8i 数据库管理系统开始了它从联机事务处理系统（OLTP）到决策支持系统和数据仓库系统的全面扩张（也许您看过一些有关 Oracle 8i 数据库管理系统的介绍，不少介绍都强调它对互连网即 Internet 的强有力的支持，但这只是它的一方面）。

决策支持系统和数据仓库系统操作的特点是：频繁地使用分组函数，如 SUM、AVG 等；频繁地使用 GROUP BY、HAVING、ORDER BY 等子句。因为对管理者和分析人员来说，他们更关心综合后的数据，但是这些操作常常要进行大规模排序，大规模排序消耗大量系统资源，特别是硬件资源。Oracle 8i 数据库管理系统为了减少排序量引入了一些新的概念和语句。

14.10　内嵌式视图

内嵌式视图（Inline Views）是一种比较复杂但却很有用的结构。实际上我们在第 10 章的最后一节中已经使用过这种结构。现在我们把第 10 章的最后一节中的例 10-60 重写，如例 14-29。

例 14-29

```
SQL> SELECT e.empno, e.ename, e.job, m.empno, m.ename, m.job
  2  FROM emp_shell e, e_m_shell, (SELECT empno, ename, job
  3                                FROM   emp) m
  4  WHERE e.empno = e_m_shell.e_id
  5  AND   e_m_shell.m_id = m.empno;
```

请注意例 14-29 查询语句中第 2 行的最后部分和第 3 行的最后部分（即阴影部分）就是内嵌式视图。

把子查询语句放在 FROM 子句中，并为该子查询语句定义一个别名来定义一个内嵌式视图。在例 14-19 的查询语句中，m 就是内嵌式视图的名字。当您定义了内嵌式视图之后，就可以像使用表或视图一样使用这个内嵌式视图。其中，视图名就是您所定义的内嵌式视图的名字，列名就是您在子查询语句 SELECT 子句中所使用的列名。

注意：

内嵌式视图不属于任何用户，也不是对象。

除了以上所讨论的用法，内嵌式视图主要用在前 n 行查询/分析（Top n queries/analysis）中。

14.11　前n行查询/分析

假设现在税务部门正在制定个人所得税的标准。因为所得税的标准是按个人收入的多少来制定的，所以税务部门的决策者们必须先知道哪些人需要纳税和纳税人的个人收入。

有统计资料表明，全世界 20%的人占有了全球 80%的资源（收入）。按照这一事实，税务部门的决策者们在制定税收政策时就要瞄着这 20%的富人，只要这部分富人的税收上来了，国家或地方的税收就有了保障（即所谓的“杀富济贫”）。而对那些穷人们，税务部门是没有办法收他们的税的，很有可能有些人连吃饭都成问题，这时政府不但不能收税而且还要发给他们最低的生活保证金。

现在的问题是如何在一个存有个人信息（包括个人收入）的表中找出这 20%的富人，您当然可以通过在查询语句中使用 ORDER BY 子句来解决这一问题，但是当这个表中存有大量的数据时，如几百万乃至几千万行数据，这样的查询语句的速度可能会慢到让人无法忍受的地步，这时，就可以使用 Oracle 8i 数据库管理系统提供的前 n 行查询/分析（Top n queries/analysis）语句来解决以上的问题。

前 n 行查询/分析不是将表中几百万乃至几千万行数据全部排序，而是使用前 n 行查询/分析（Top n queries/analysis）语句只将 20%的富人的数据进行排序，这样系统的排序量就会大大地减少，因而查询语句的速度也就明显提高。

可使用前 n 行查询/分析（Top n queries/analysis）语句的地方有：

- 本地区 200 强企业（纳税最高的）。
- 本地区最贫困的 1000 人的个人收入。
- 本地区最老的 10 名老寿星。
- 本公司最出色的前 5 名推销员（销售额最高的）。
- 本公司销售最好的前 10 种商品等。

前 n 行查询/分析（Top n queries/analysis）语句的格式如下：

```
SELECT ROWNUM,[列名[, 列名…]]
FROM   ([列名[, 列名…]]
         FROM   表名
         ORDER BY "Top n 列名"
WHERE  ROWNUM <= N;
```

子查询或内嵌式视图（Inline Views）将产生一个数据的存储列表。为了保证存储列表中的数据按所需要的顺序排列，要在子查询或内嵌式视图（Inline Views）中包含 ORDER BY 子句。如果您想得到最大的几行数据，您必须在 ORDER BY 子句中使用 DESC 关键字。

在外层查询中限制最终结果所包含的数据行数。在外层查询中，ROWNUM 是一伪列（即它不是一个真实的列，但是您可以把它当作列来使用）。ROWNUM 是子查询或内嵌式视图（Inline Views）所返回的序列号，该序列号从 1 开始。WHERE 子句返回数据行数（n 行）。在外层查询语句中必须包含“<”或“<=”操作符。

假设公司现在要进一步节省开销，人员的工资是公司的主要开销之一。您的老总想知道公司中工资最高的 5 个人（当然不能包含他本人），他让您找出公司中工资最高的 5 个人的清单，他要好好地研究。您使用了例 14-30 的查询语句来完成这个工作。

例 14-30

```
SQL> SELECT rownum "Order NO.", ename "Name", sal "Salary", job "Job"
  2  FROM  (SELECT ename,sal,job
```

```
        FROM emp
        WHERE job NOT LIKE 'PRESI%'
        ORDER BY sal DESC)
  WHERE ROWNUM <= 5;
```

例 14-30 结果

```
Order NO. Name          Salary Job
-------- --------- ---------- ------
       1 SCOTT            3000   ANALYST
       2 FORD             3000   ANALYST
       3 JONES            2975   MANAGER
       4 BLAKE            2850   MANAGER
       5 CLARK            2450   MANAGER
```

假设您的公司是一个大型的跨国企业，emp 表中存有一百多万行数据（即一百多万名员工），如果您不像例 14-30 那样使用前 n 行查询/分析（Top n queries/analysis）语句来查询的话，这个语句可能要执行很长时间。

当您很快地把例 14-30 显示的结果交给您的老总时，他感到很惊讶，他对您的工作效率非常满意。也许心里在想是不是他叫您干的工作把您吓坏了，所以干活才这么卖命。看来以后时常得让手下的人有点危机感，这样他们干活才不敢偷懒。

14.12　ROWNUM的更多应用

假设公司中的一个表中有几千万行数据，而普通用户可以通过互联网来浏览这个表。许多用户最初只是想看看这个表中是否有他们需要的信息，因此随便看到一些数据就可以。当然我们使用 ORDER BY 和 WHERE 子句等传统方法可以获得这些数据，但是 SQL 语句相对比较复杂，也可能会产生严重的效率问题。此时使用 ROWNUM 伪列就可以方便而高效地完成这类用户的需求。

为了让读者有所体会，我们选一个有九十多万行数据的表。为此，首先启动 SQL*Plus 并以 SYSTEM 用户登录数据库，然后使用例 14-31 的命令将 sh 用户解锁（如果该用户已经可以使用，这一步可以省略）。

例 14-31

```
SQL> alter user sh identified by sh account unlock;
```

例 14-31 结果

```
用户已更改。
```

然后使用例 14-32 的 connect 命令切换到用户 sh。

例 14-32

```
SQL> connect sh/sh
```

例 14-32 结果

```
已连接。
```

使用了例 14-33 的带有分组函数 COUNT 的查询语句来获取 sales 表的数据行的总数。

例 14-33

```
SQL> select count(*) from sales;
```

例 14-33 结果

```
 COUNT(*)
----------
   918843
```

从例 14-33 的显示结果可以看出，sales 表中有 918843 行数据，这正是我们所需要的。使用例 14-34 的 SQL*Plus 命令获取 sales 表结构。

例 14-34

```
SQL> desc sales
```

例 14-34 结果

```
 名称                                      是否为空? 类型
 ----------------------------------- -------- ------------
 PROD_ID                                   NOT NULL NUMBER
 CUST_ID                                   NOT NULL NUMBER
 TIME_ID                                   NOT NULL DATE
 CHANNEL_ID                                NOT NULL NUMBER
 PROMO_ID                                  NOT NULL NUMBER
 QUANTITY_SOLD                             NOT NULL NUMBER(10,2)
 AMOUNT_SOLD                               NOT NULL NUMBER(10,2)
```

接下来使用例 14-35 的 SQL 查询语句显示 sales 表中的 12 行数据（也可以显示更多或更少行）。

例 14-35

```
SQL> select *
  2  from sales
  3  where rownum <= 12;
```

例 14-35 结果

```
   PROD_ID  CUST_ID TIME_ID    CHANNEL_ID  PROMO_ID  QUANTITY_SOLD AMOUNT_SOLD
--------- -------- ---------- ---------- ----  ----------------- -----------
       13      987 10-1月 -98          3       999               1     1232.16
       13     1660 10-1月 -98          3       999               1     1232.16
       13     1762 10-1月 -98          3       999               1     1232.16
       13     1843 10-1月 -98          3       999               1     1232.16
       13     1948 10-1月 -98          3       999               1     1232.16
```

```
        13       2273 10-1 月 -98           3          999              1    1232.16
        13       2380 10-1 月 -98           3          999              1    1232.16
        13       2683 10-1 月 -98           3          999              1    1232.16
        13       2865 10-1 月 -98           3          999              1    1232.16
        13       4663 10-1 月 -98           3          999              1    1232.16
        13       5203 10-1 月 -98           3          999              1    1232.16
        13       5321 10-1 月 -98           3          999              1    1232.16

已选择 12 行。
```

您可以发现我们不但得到了所需数据，而且速度非常快，这正是使用 Oracle 引入 ROWNUM 伪列的又一方便之处。

14.13　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 什么是视图（Views）？
- 引入视图（Views）的原因。
- 什么是数据的独立性？
- 如何利用视图（Views）来维护数据的独立性？
- 使用视图的主要好处。
- 如何创建视图（Views）？
- 如何修改视图（Views）？
- 使用视图对系统效率的冲击。
- 如何利用 Oracle 数据字典来查看有关视图（Views）的信息？
- 如何使用视图（Views）来进行 DML 操作？
- 如何使用 WITH CHECK OPTION 子句？
- 使用 WITH READ ONLY 子句的原因。
- 如何使用 WITH READ ONLY 子句？
- 如何删除视图（Views）？
- 什么是内嵌式视图（Inline Views）？
- 如何使用内嵌式视图（Inline Views）？
- 前 n 行查询/分析（Top n queries/analysis）语句引入的原因。
- 什么是前 n 行查询/分析（Top n queries/analysis）语句？
- 如何使用前 n 行查询/分析（Top n queries/analysis）语句？

第15章

序列号和同义词

在前面的几章中我们已经介绍了表、索引和视图 3 种常用的对象，下面我们介绍另外两种对象——序列号（Sequence）和同义词（Synonym）。

15.1 序列号的引入

假设您的公司是一个大型企业，每天都要向许多不同的供应商订货，从而产生许多订单。一般公司要求这些订单号要连续，即每开一张订单其订单号码加 1。如果采用手工的方式就要求开订单的人每次手工地输入订单号码，这样很容易出错而且也增加了工作量。

进一步假设在您的公司中有十多个员工有权发订单，他们并不是工作在同一个办公室中，有的可能还不在同一栋建筑中，甚至可能不在同一个城市内。在这种情形下要想让他们产生号码连续的订单几乎是不可能的。

要解决这一难题，您可以自己使用程序设计语言开发一个自动生成订单号码的程序。虽然这是一种解决这一难题的方法，但是这无疑增加了信息系统开发的成本。另外，这一程序代码的质量完全取决于程序员的水平。Oracle 又一次高瞻远瞩地引入了序列号（Sequence）来帮助您轻松地解决了以上难题。

Oracle 的序列号（Sequence）有如下的特点：

- 它是一个可以由多个用户共享的数据库对象。
- 它是一个序列号产生器，专为表中的数据行自动产生序列号。
- 序列号是由 Oracle 内部例程产生和维护的。
- 它独立于使用它的表。
- 它通常是用来产生主键（唯一的号码）。
- 它可以取代产生序列号的应用程序。
- 如果让它常驻内存，可提高访问序列号的效率。

15.2 创建序列号语句的格式

您使用 CREATE SEQUENCE 语句来创建序列号（Sequence）。CREATE SEQUENCE 语句也是一个 DDL 语句。

CREATE SEQUENCE 语句的格式如下：

```
CREATE SEQUENCE　序列号的名字
	[START WITH n]
	[INCREMENT BY n]
	[{MAXVALUE n | NOMAXVALUE}
	[{MINVALUE n | NOMINVALUE}]
	[{CACHE n | NOCACHE}]
	[{CYCLE n | NOCYCLE | CYCLE 20}];
```

其中各参数的含义如下。

- 序列号的名字：为序列号产生器的名字。
- START WITH n：定义了所产生的第 1 个序列号码，这里 n 为整数。如果该子句省略，序列号从 1 开始。
- INCREMENT BY n：定义了序列号增加步长（序列号之间的间隔）。如果该子句省略，序列号增加步长（序列号之间的间隔）为 1。
- MAXVALUE n：定义了可产生的序列号的最大值。
- NOMAXVALUE：说明升序的序列号的最大值为 10 的 27 次方，而降序的序列号的最大值为-1（这也是默认值）。
- MINVALUE n：定义了可产生的序列号的最小值。
- NOMINVALUE：说明升序的序列号的最小值为 1，而降序的序列号的最小值为-10 的 26 次方（这也是默认值）。
- CACHE n：说明将有 n 个序列号码被 Oracle 服务器预分配和保存在内存中。
- NOCACHE：说明将没有序列号被 Oracle 服务器预分配和保存在内存中。
- CACHE 20：这是默认值，您不用说明。
- CYCLE n：说明在序列号达到最大值或最小值后，将继续产生序列号。
- NOCYCLE：说明在序列号达到最大值或最小值后，将不再产生序列号（这也是默认值）。

15.3　如何创建序列号

还记得在第 10 章的例 10-11 中所创建的表 supplier 吗？您可以使用例 15-1 的 SQL*Plus 命令来查看这个表的结构。

例 15-1

```
SQL> desc supplier
```

例 15-1 结果

```
名称                                       是否为空？ 类型
 --------------------------------- --------- ------------
 S_CODE                                               NUMBER(6)
 SNAME                                                VARCHAR2(25)
```

```
CONTACT                                             VARCHAR2(15)
PHONE                                               VARCHAR2(15)
FAX                                                 VARCHAR2(15)
```

您还应该使用例 15-2 的查询语句来查看 supplier 表中所存的数据。

例 15-2

```
SQL> select * from supplier;
```

例 15-2 结果

```
未选定行
```

例 15-2 显示的结果表明，supplier 表中没有任何数据。这也是在预料之中的，因为从这个表建立以来还没人往里面输入数据。

为了让 Oracle 系统自动产生供应商号（s_code），您可以使用例 15-3 的 DDL 语句来创建一个叫 supplier_s_code 的序列号（产生器）。

例 15-3

```
SQL> CREATE SEQUENCE supplier_s_code
  2     START WITH 2000
  3     INCREMENT BY 10
  4     MAXVALUE 100000
  5     NOCACHE
  6     NOCYCLE;
```

例 15-3 结果

```
序列已创建。
```

例 15-3 的结果只显示“序列已创建。”从这个结果中您是无法得知一个序列号（产生器）的详细信息的，可以通过使用 Oracle 提供的数据字典 user_sequences 来得到有关的信息。

现在可以使用例 15-5 的查询语句来检查您刚创建的序列号（产生器）是否正确。不过在这之前您应该使用例 15-4 的 SQL*Plus 命令将 sequence_name 重新格式化一下。

例 15-4

```
SQL> col sequence_name for a18
```

例 15-5

```
SQL> SELECT sequence_name, min_value, max_value,
  2         increment_by, last_number
  3  FROM user_sequences;
```

例 15-5 结果

```
SEQUENCE_NAME       MIN_VALUE  MAX_VALUE INCREMENT_BY LAST_NUMBER
------------------ ---------- ---------- ------------ -----------
SUPPLIER_S_CODE             1     100000           10        2000
```

例 15-5 所显示的结果表明，您刚创建的序列号（产生器）supplier_s_code 已准确地记录到 Oracle 系统中。

您也可以使用数据字典 user_objects 来得到有关序列号（产生器）supplier_s_code 的信息，如也可以使用例 15-6 的查询语句。

例 15-6

```
SQL> SELECT object_name, object_type, created,
  2         last_ddl_time, status
  3  FROM user_objects
  4  WHERE object_type != 'TABLE'
  5  AND   object_type != 'INDEX';
```

例 15-6 结果

```
OBJECT_NAME                 OBJECT_TYPE          CREATED     LAST_DDL_T  STATUS
------------------- ------------------- ---------- ---------- ------
AVERAGE                     VIEW                  15-4 月 -03 15-4 月 -03 VALID
SALARY_LIMIT                VIEW                  15-4 月 -03 15-4 月 -03 VALID
SALES30                     VIEW                  15-4 月 -03 15-4 月 -03 VALID
SUPPLIER_S_CODE             SEQUENCE              26-4 月 -03 26-4 月 -03 VALID
```

现在您就可以使用这个序列号（产生器）了。

15.4 如何使用创建的序列号

那么我们如何利用这个序列号向表中插入数据呢？

还记得我们在第 1 章中讲过的最基本的 SELECT 语句吗？SELECT 语句必须至少包含两个子句，即 SELECT 子句和 FROM 子句。也许是为了能利用 SELECT 语句得到序列号的相关信息，Oracle 引入了两个虚（伪）列，既然叫虚（伪）列，它们就不是表中存在的真实的列，但您可以像使用真实的列一样去使用这些伪列。

两个虚（伪）列是 NEXTVAL 和 CURRVAL，从这两个虚（伪）列的名字看，您应该可以大概猜出它们的基本含义或用处了吧。

- NEXTVAL 用于返回序列号的下一个可获得的值。
- CURRVAL 用于获得序列号的当前值。

现在您可以试着利用虚表 dual 来获得序列号 supplier_s_code 的当前值。您可以试着使用例 15-7 的查询语句。

例 15-7

```
SQL> SELECT supplier_s_code.CURRVAL
  2  FROM dual;
```

例 15-7 结果

```
SELECT supplier_s_code.CURRVAL
       *
ERROR 位于第 1 行:
ORA-08002: 序列 SUPPLIER_S_CODE.CURRVAL 尚未在此进程中定义
```

看到例 15-7 显示的结果您可能会感到困惑，因为在例 15-7 所使用的 SQL 语句中找不到任何语法错误，这是为什么呢？

Oracle 规定：在引用 CURRVAL 之前，必须在当前的会话中（账号下）使用 NEXTVAL 产生一个序列号的值。

现在我们暂时搁置这方面的讨论，先利用序列号 supplier_s_code 向表 supplier 中插入一行数据。您可以使用例 15-8 的 DML 语句来完成这一工作（请注意 supplier_s_code. NEXTVAL 的用法）。

例 15-8

```
SQL> INSERT INTO supplier(s_code, sname, contact, phone, fax)
  2  VALUES              (supplier_s_code.NEXTVAL, '仙客来百货',
     3                       '张根发', 4444944, 4444844);
```

例 15-8 结果

```
已创建 1 行。
```

现在您应该使用例 15-10 的查询语句来检查刚做的插入操作是否准确无误。但是在执行这一查询语句之前最好使用例 15-9 的 SQL*Plus 格式化命令以得到清晰的输出显示。

例 15-9

```
SQL> col sname for a16
SQL> col contact for a12
SQL> col phone for a10
SQL> col fax for a10
```

例 15-10

```
SQL> SELECT *
    2  FROM supplier;
```

例 15-10 结果

```
    S_CODE  SNAME             CONTACT      PHONE      FAX
---------- ---------------- ------------ ---------- ---------
      2000  仙客来百货         张根发        4444944    4444844
```

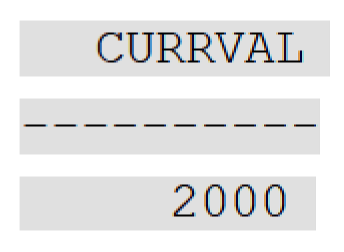
看到了例 15-10 的结果您可能会很兴奋，因为您已能熟练地运用序列号向表中插入数据了。

现在您可以再试着利用虚表 dual 来获得序列号 supplier_s_code 的当前值，可以使用例 15-11 的查询语句。

例 15-11

```
SQL> SELECT supplier_s_code.CURRVAL
    2  FROM dual;
```

例 15-11 结果



```
   CURRVAL
----------
      2000
```

这次您终于利用查询语句获得了序列号 supplier_s_code 的当前值。

为了进一步加深对 NEXTVAL 和 CURRVAL 这两个虚（伪）列的理解，您可以再使用例 15-12 的 DML 语句向表 supplier 中插入一行数据。

例 15-12

```
SQL> INSERT INTO supplier(s_code, sname, contact, phone, fax)
  2  VALUES               (supplier_s_code.NEXTVAL, '心太软小商品',
  3                       '石铁心', 1741741, 1741742);
```

例 15-12 结果

```
已创建 1 行。
```

然后，您应该使用例 15-13 的查询语句来检查刚做的插入操作是否正确。

例 15-13

```
SQL> SELECT *
    2  FROM supplier;
```

例 15-13 结果

```
    S_CODE SNAME                CONTACT         PHONE       FAX
---------- ---------------- ------------- ----------- --------
      2000 仙客来百货           张根发          4444944     4444844
      2010 心太软小商品         石铁心          1741741     1741742
```

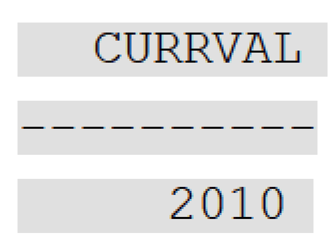
当看到例 15-13 显示的结果时，您可能也为自己驾驭 Oracle 的能力感到自豪。

现在您可以再试着利用虚表 dual 来获得序列号 supplier_s_code 的当前值，可以使用例 15-14 的查询语句。

例 15-14

```
SQL> SELECT supplier_s_code.CURRVAL
    2  FROM dual;
```

例 15-14 结果



```
   CURRVAL
----------
      2010
```

现在您应该基本上理解了 NEXTVAL 和 CURRVAL 这两个虚（伪）列的含义和用法了吧！

15.5　使用序列号的实例

在商业运作中，当某一公司向它的供应商订货时，一般一张订单都包括多个商品在这样的订单上，订单号和供应商号是不变的（当然订货日期也是不变的）。现在的问题是怎样利用刚学过的序列号向订单（ord 表）中插入这样的数据呢？

首先，您应使用例 15-15 的 SQL*Plus 命令来查看 ord 表的结构。

例 15-15

```
SQL> DESC ord
```

例 15-15 结果

```
名称                                      是否为空? 类型
----------------------------------------- -------- ------------
ORDNO                                              NUMBER(8)
P_CODE                                             NUMBER(6)
S_CODE                                             NUMBER(6)
ORDATE                                             DATE
UNIT                                               NUMBER(6)
PRICE                                              NUMBER(8,2)
```

注意：

如果您还没有建立这个表，现在您应该使用 CREATE TABLE 语句创建一个具有如上结构的 ord 表。

为了让 Oracle 系统自动产生订单号（ordno），您可以使用例 15-16 的 DDL 语句来创建一个叫 ord_ordno 的序列号（产生器）。

例 15-16

```
SQL> CREATE SEQUENCE ord_ordno
  2  START WITH 1000
  3  INCREMENT BY 1
  4  MAXVALUE 100000
  5  NOCYCLE;
```

例 15-16 结果

```
序列已创建。
```

虽然例 15-16 的结果已显示“序列已创建。”，但从这个结果中您无法得知这个序列号的详细信息。因此，您可以通过使用 Oracle 提供的数据字典 user_sequences 来得到有关的信息。

现在您应该使用例 15-17 的查询语句来检查刚创建的序列号是否正确。

例 15-17

```
SQL> SELECT sequence_name, min_value, max_value,
  2         increment_by, last_number, cache_size
  3  FROM user_sequences;
```

例 15-17 结果

```
SEQUENCE_NAME      MIN_VALUE  MAX_VALUE INCREMENT_BY LAST_NUMBER CACHE_SIZE
------ ---------- ---------- ------------ ----------- ----------
ORD_ORDNO                 1     100000            1        1000         20
SUPPLIER_S_CODE            1     100000           10        2020          0
```

从例 15-17 结果中的 CACHE_SIZE 显示，您可以看出使用 NOCACHE 子句和不使用 NOCACHE 子句的区别。因为当您使用例 15-3 的 DDL 语句来创建 supplier_s_code 序列号时，您使用了 NOCACHE 子句，所以例 15-17 显示结果中 supplier_s_code 的 CACHE_SIZE 列为 0。但是当您使用例 15-16 的 DDL 语句来创建 ord_ordno 序列号时，您没有使用 NOCACHE 子句，所以例 15-17 显示结果中 ord_ordno 的 CACHE_SIZE 列为 20（这也是默认值）。

现在您就可以利用序列号 ord_ordno 往表 ord 中插入一行数据。您可以使用例 15-18 的 DML 语句来完成这一工作。

例 15-18

```
SQL> INSERT INTO ord(ordno, p_code, s_code, ordate, unit, price)
  2  VALUES        (ord_ordno.NEXTVAL, 881, supplier_s_code.CURRVAL,
  3               SYSDATE, 10, 2.5);
```

例 15-18 结果

```
已创建 1 行。
```

虽然例 15-18 显示的结果为“已创建 1 行。”，但是为了慎重起见，您还是应该使用例 15-19 的查询语句来检查刚做的插入操作是否准确无误。

例 15-19

```
SQL> SELECT *
  2  FROM ord;
```

例 15-19 结果

```
     ORDNO     P_CODE     S_CODE ORDATE              UNIT      PRICE
---------- ---------- ---------- ---------- ---------- ----------
      1000        881       2010 27-4 月 -03            10        2.5
```

现在您就可以使用例 15-20 的 DML 语句再向 ord 中插入一行订单号、供应商号和订单日期（ORDNO、S_CODE 和 ORDATE）与上一行完全相同，但其他列不一样的数据。

例 15-20

```
SQL> INSERT INTO ord(ordno, p_code, s_code, ordate, unit, price)
  2  VALUES        (ord_ordno.CURRVAL, 882, supplier_s_code.CURRVAL,
  3               SYSDATE, 15, 6.12);
```

例 15-20 结果

```
已创建 1 行。
```

虽然例 15-20 显示的结果为“已创建 1 行。”，但是为了慎重起见，您还是应该使用例 15-21 的查询语句来检查刚做的插入操作是否准确无误。

例 15-21

```
SQL> SELECT *
  2  FROM ord;
```

例 15-21 结果

```
    ORDNO      P_CODE      S_CODE  ORDATE              UNIT        PRICE
---------- ---------- ---------- ---------- ---------- ----------
      1000         881       2010  27-4 月 -03           10          2.5
      1000         882       2010  27-4 月 -03           15         6.12
```

例 15-21 显示结果表明，您已经成功地向 ord 表中插入两行订单号、供应商号和订货日期都不变的数据。如果需要的话，您可以使用与例 15-20 相似的 DML 语句插入任意多行类似的数据。

注意：

在这样的 DML 语句中插入数据时，要使用伪列 CURRVAL，而不能使用 NEXTVAL，因为使用 NEXTVAL 后会造成所引用的序列号的值变化。

如果您仔细地观察一下刚创建的两个序列号，您就会注意到在创建它们时都用了 NOCYCLE 子句（这也是默认值）。虽然 Oracle 允许您在创建序列号时使用 CYCLE 子句，但是您应该尽量少用，特别是用序列号来产生主键时。如果您不得不用的话，您必须建立一种清除旧的主键的机制，从而使旧的主键清除的速度比循环（CYCLE）的速度快。

15.6 NEXTVAL和CURRVAL虚（伪）列介绍和它们的使用规则

当创建了一个序列号之后，就可以使用这个序列号为表中的数据行产生顺序的号码。序列号是通过使用伪列 NEXTVAL 和 CURRVAL 来引用的。

NEXTVAL 和 CURRVAL 有如下要注意的特点：

- 利用伪列 NEXTVAL 来得到所定义的序列号（产生器）的下一个序列号码。
- 在使用伪列 NEXTVAL 时，必须在 NEXTVAL 之前冠以所定义的序列号。
- 当引用序列号.NEXTVAL 时，将产生一个新的序列号码，并且这个新的序列号码被放到 CURRVAL 伪列中。
- 用伪列 CURRVAL 来引用刚产生的序列号码。
- 在引用 CURRVAL 之前，您一定在当前的会话（账号）中使用 NEXTVAL 产生一个序列号码。

您可以使用 NEXTVAL 和 CURRVAL 的情况如下：

- 在查询语句的 SELECT 子句中，但不包括子查询中的 SELECT 子句。
- 在 UPDATE 语句的 SET 子句中。
- 在 INSERT 语句的子查询的 SELECT 列表中。
- 在 INSERT 语句的 VALUES 子句中。

您不能使用 NEXTVAL 和 CURRVAL 的情况如下：

- 在视图的 SELECT 子句中。
- 在查询（SELECT）语句的子查询中。
- 在 UPDATE 语句的子查询中。
- 在 DELETE 语句的子查询中。
- 在包含 DISTINCT 关键字的查询（SELECT）语句中。
- 在包含 ORDER BY 子句的查询（SELECT）语句中。
- 在包含 GROUP BY 子句的查询（SELECT）语句中。
- 在包含 HAVING 子句的查询（SELECT）语句中。
- 在包含 DEFAULT 表达式的 CREATE TABLE 语句中。
- 在包含 DEFAULT 表达式的 ALTER TABLE 语句中。

如果某个序列号是多个用户共享的，而且访问的频率很高，为了加快访问该序列号的速度，您可以将它存在内存中，但是这样做的副作用是序列号码有可能产生含有间隔的序列号码。

利用 CREATE SEQUENCE 语句的 CACHE n 子句来说明某个序列号前 n 个值将被放入内存中。Oracle 服务器的操作包括：

（1）在第 1 次引用这个序列号时，Oracle 服务器计算出 n 个序列号码并将它们放入内存。

（2）每一个要求下一个序列号码的请求都从在内存中的序列号码得到。

（3）当内存中的最后一个序列号码用完后而又收到下一个请求时，Oracle 服务器再计算出 n 个序列号码并将它们放入内存。

下面我们通过一个例子来进一步说明这一问题。首先您要使用 SCOTT/TIGER 重新登录 SQL*Plus，然后可以使用例 15-22 的查询语句来查看您所建的序列号的相关信息。

例 15-22

```
SQL> SELECT sequence_name, min_value, max_value,
  2         increment_by, last_number, cache_size
  3  FROM user_sequences;
```

例 15-22 结果

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER	CACHE_SIZE
ORD_ORDNO	1	100000	1	1020	20
SUPPLIER_S_CODE	1	100000	10	2020	0

请注意例 15-22 结果中的 LAST_NUMBER 这一列，ORD_ORDNO 的这一列的值为 1020。这个值与例 15-21 的查询语句中的 ORDNO 的值（1000）存在 20 个数的间隔。这是因为虽然您只使用了一个序列号码 1000，但是 Oracle 服务器却已经算出了 20 个序列号码并把它们都放在了内存中，所以当您退出 SQL*Plus 再重新登录后就产生了这一间隔。

现在您可以试着使用序列号 ord_ordno 向 dept_dml 表中插入一行数据，如例 15-23 的 DML 语句。

例 15-23

```
SQL> INSERT INTO dept_dml(deptno, dname, loc)
  2  VALUES           (ord_ordno.NEXTVAL, '培训部', '待定');
```

例 15-23 结果

```
VALUES           (ord_ordno.NEXTVAL, '培训部', '待定')
                  *
ERROR 位于第 2 行:
ORA-01438: 值大于此列指定的允许精确度
```

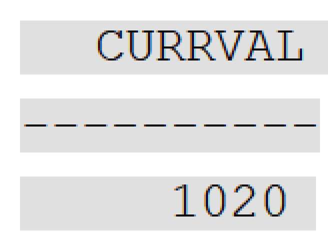
例 15-23 显示的结果表明，例 15-23 的 DML 语句中的插入操作失败了。您能猜出 Oracle 是如何处理序列号 ord_ordno 的吗？

为此，您可以使用例 15-24 的查询语句来得到序列号 ord_ordno 的当前值。

例 15-24

```
SQL> SELECT ord_ordno.CURRVAL
  2  FROM dual;
```

例 15-24 结果



```
  CURRVAL
----------
      1020
```

看到了例 15-24 显示的结果您应该知道了，虽然您的 SQL 语句是错的，但因为 deptno 的最大长度为两位整数（即 0~99），但是 Oracle 系统并没有回滚该语句所操作的序列号。

如果您使用了例 15-25 的查询语句，您会更清楚地理解这一点。

例 15-25

```
SQL> SELECT sequence_name, min_value, max_value,
  2         increment_by, last_number, cache_size
  3  FROM user_sequences;
```

例 15-25 结果

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER	CACHE_SIZE
ORD_ORDNO	1	100000	1	1040	20
SUPPLIER_S_CODE	1	100000	10	2020	0

请注意例 15-25 结果中的 LAST_NUMBER 这一列中的 ORD_ORDNO 一行的值已经增加为 1040。

如果某个序列号(Sequence)是以 NOCACHE 方式(即 user_sequences 中的 CACHE_SIZE 列为 0）建立的，您可以在不增加序列号码的情况下使用数据字典 user_sequences 查看它的下一个值（LAST_NUMBER 列的值）。

虽然序列号产生器产生的序列号码本身是没有间隔的，但是这一操作是独立于提交或

回滚操作（COMMIT 或 ROLLBACK）的，而且序列号本身又是一个可共享的对象，因此，在下列情况下有可能造成序列号码的间隔（序列号码的丢失）：

- 当使用该序列号的事务被回滚时。
- 当该序列号还用于另外的表时。
- 当系统崩溃时。

15.7　序列号的修改

商业环境是不断变化的，所以在某些情况下已定义的序列号（Sequence）也需进行相应的调整。Oracle 总是高瞻远瞩，它早就预见到了这一点，而且提供了 ALTER SEQUENCE 语句。假设您发现序列号 SUPPLIER_S_CODE 的一些参数已经不适用了，您可以用例 15-26 的 DDL 语句来修改这一序列号。

例 15-26

```
SQL> ALTER SEQUENCE supplier_s_code
  2  INCREMENT BY 11
  3  MAXVALUE 999999
  4  CACHE 50;
```

例 15-26 结果

```
序列已更改。
```

现在您可以通过数据字典 user_sequences 查看所做的修改是否已记录到系统中，如例 15-27 所示。

例 15-27

```
SQL> SELECT sequence_name, min_value, max_value,
  2         increment_by, last_number, cache_size
  3  FROM user_sequences;
```

例 15-27 结果

```
SEQUENCE_NAME        MIN_VALUE  MAX_VALUE INCREMENT_BY LAST_NUMBER CACHE_SIZE
----- ---------- ---------- ------------ ----------- ------------------
ORD_ORDNO                    1     100000            1        1040         20
SUPPLIER_S_CODE              1     999999           11        2021         50
```

从例 15-27 的结果可以看出，您对序列号 SUPPLIER_S_CODE 所做的修改已被准确无误地记录到系统中。

您可能会问：“当修改了序列号 SUPPLIER_S_CODE 之后，Oracle 系统该如何处理曾利用该序列号插入数据行中的相应的序列号码呢？”

为了回答这一问题，现在您可以使用例 15-28 的 INSERT 语句，利用虚列 supplier_s_code.NEXTVAL 向 supplier 表中插入一行新数据。

例 15-28

```
SQL> INSERT INTO supplier(s_code, sname, contact, phone, fax)
  2  VALUES               (supplier_s_code.NEXTVAL, '食为天餐具',
  3                       '金元宝', 1671671, 1671674);
```

例 15-28 结果

```
已创建 1 行。
```

然后，您可以使用例 15-29 的查询语句来检查 supplier 表中的变化。

例 15-29

```
SQL> SELECT *
  2  FROM supplier;
```

例 15-29 结果

```
    S_CODE  SNAME                        CONTACT         PHONE            FAX
----------  ----------------  ----------------  ----------------  --------
      2000  仙客来百货                   张根发          4444944          4444844
      2010  心太软小商品                 石铁心          1741741          1741742
      2021  食为天餐具                   金元宝          1671671          1671674
```

从例 15-29 显示的结果可以看出，对于序列号的修改并不影响修改之前的数据，而只影响修改之后的数据。

您也可以使用例 15-30 的查询语句来查看序列号 SUPPLIER_S_CODE 的当前值。

例 15-30

```
SQL> SELECT supplier_s_code.CURRVAL
  2  FROM dual;
```

例 15-30 结果

```
   CURRVAL
----------
      2021
```

您也可以使用例 15-31 的查询语句来查看序列号 SUPPLIER_S_CODE 有什么变化。

例 15-31

```
SQL> SELECT sequence_name, min_value, max_value,
  2         increment_by, last_number, cache_size
  3  FROM user_sequences;
```

例 15-31 结果

```
SEQUENCE_NAME       MIN_VALUE   MAX_VALUE INCREMENT_BY LAST_NUMBER CACHE_SIZE
------ ---------- ---------- ------------ ----------- -----------------
ORD_ORDNO                   1      100000            1        1040         20
SUPPLIER_S_CODE             1      999999           11        2571         50
```

在例 15-31 的结果中，LAST_NUMBER 已由 2021 增加到 2571。您知道这是为什么吗？因为 2021+11×50=2021+550=2571。

为了进一步理解序列号的操作，您可以再使用例 15-32 的 INSERT 语句，利用虚列 supplier_s_code.NEXTVAL 向 supplier 表中插入一行新数据。

例 15-32

```
SQL> INSERT INTO supplier(s_code, sname, contact, phone, fax)
  2  VALUES            (supplier_s_code.NEXTVAL, '食为先餐具',
  3                    '陆合彩', 1681684, 1681684);
```

例 15-32 结果

```
已创建 1 行。
```

现在您再使用例 15-33 的查询语句来检查 supplier 表中的变化。

例 15-33

```
SQL> SELECT *
  2  FROM supplier;
```

例 15-33 结果

```
   S_CODE  SNAME                     CONTACT        PHONE          FAX
---------- --------------- --------------- --------------- -------
      2000  仙客来百货                张根发         4444944        4444844
      2010  心太软小商品              石铁心         1741741        1741742
      2021  食为天餐具                金元宝         1671671        1671674
      2032  食为先餐具                陆合彩         1681684        1681684
```

现在您应该对序列号的使用更加了解了。

ALTER SEQUENCE 序列号语句的格式如下：

ALTER SEQUENCE　序列号的名字

[INCREMENT BY n]

[{MAXVALUE n | <u>NOMAXVALUE</u>}

[{MINVALUE n | <u>NOMINVALUE</u>}]

[{CACHE n | <u>NOCACHE</u>}]

[{CYCLE n | NOCYCLE}];

下面是一些在修改序列号时需要注意的事项：

- 您必须是序列号的拥有者或具有 ALTER ANY SEQUENCE 的系统权限。
- 您不能修改 START WITH 选项，您只能通过将该序列号删除，然后再重建来达到这一目的。
- ALTER SEQUENCE 只影响以后的序列号码。
- 在做修改之前，Oracle 系统要执行一些检查，如修改后的 MAXVALUE 不能小于序列号的当前值。

15.8 删除序列号

一旦某一序列号（Sequence）不再需要时，您就可以使用 DROP SEQUENCE 语句将其删除。例如，您可以使用例 15-34 的 DDL 语句将序列号 supplier_s_code 删除。

例 15-34

```
SQL> DROP SEQUENCE supplier_s_code;
```

例 15-34 结果

```
序列已丢弃。
```

您可以使用例 15-35 的查询语句来检查是否已成功地删除了序列号 supplier_s_code。

例 15-35

```
SQL> SELECT sequence_name, min_value, max_value,
  2        increment_by, last_number, cache_size
  3  FROM user_sequences;
```

例 15-35 结果

```
SEQUENCE_NAME   MIN_VALUE  MAX_VALUE INCREMENT_BY LAST_NUMBER CACHE_SIZE
------- ---------- ---------- ------------ ----------- ------------------
ORD_ORDNO               1     100000            1        1040         20
```

从例 15-35 的结果可以看出，序列号 supplier_s_code 已被成功地删除。

虽然 DROP SEQUENCE supplier_s_code 语句已经把序列号 supplier_s_code 从 Oracle 系统中删除了，但是使用序列号 supplier_s_code 插入的数据依旧存在。您可以使用例 15-36 的查询语句来验证这一点。

例 15-36

```
SQL> SELECT *
  2  FROM supplier;
```

例 15-36 结果

```
   S_CODE  SNAME                CONTACT         PHONE           FAX
---------- ---------------- --------------- --------------- -------
     2000  仙客来百货            张根发           4444944         4444844
     2010  心太软小商品          石铁心           1741741         1741742
     2021  食为天餐具            金元宝           1671671         1671674
     2032  食为先餐具            陆合彩           1681684         1681684
```

一旦您删除了某一序列号，就不能再引用它了，您可以使用例 15-37 的 DML 语句来验证这一点。

例 15-37

```
SQL> INSERT INTO supplier(s_code, sname, contact, phone, fax)
  2  VALUES              (supplier_s_code.NEXTVAL, '环宇批发',
  3                      '', 1234567, 1234567);
```

例 15-37 结果

```
VALUES              (supplier_s_code.NEXTVAL, '环宇批发',
                    *
ERROR 位于第 2 行:
ORA-02289: 序列（号）不存在
```

删除序列号语句的格式如下：

DROP SEQUENCE　序列号的名;

在结束序列号的讨论之前，再简单介绍序列号在商业运作时可能要注意的一个问题。如果您仔细地回忆一下我们在这一章所创建的两个序列号的起始值，没有一个是从 1 开始的（START WITH 1），如果您曾注意过商业订单或发票等的话，也会发现很少有从 1 或 0 开始的。您能说出为什么吗？

因为一般商业公司或机构都希望向外界传递一种生意蒸蒸日上的信息，这种信息反映在订单或发票上就是号码很大，从而使客户或供应商认为该公司的生意太繁忙了。

设想一下，有一位美丽的小姐为了更加美丽而到一所美容医院去做美容手术，主持手术的医生对这位小姐说："这是我大学毕业的第一份工作，也是我第一次给人做手术，我一定百倍地珍惜这次难得的机会，尽我最大的努力把您的手术做好。"我不知道这位美丽的小姐还有没有勇气让这位从来没拿过手术刀的家伙在她漂亮的脸蛋上练手。

15.9　引入同义词的原因

在一些商业数据库中，有时信息系统的设计者或开发者为了增加信息的易读性，故意定义一些很长的表名（也可能是其他的对象）。这样虽然增加了易读性，但在引用这些表或对象时就不那么方便，也容易产生输入错误。另外，在实际的商业公司里，一些用户觉得某一个对象名有意义也很好记，但另一些用户可能觉得另一个名字更有意义。

Oracle 系统提供的同义词（Synonym）就是用来解决以上难题的。设想一下在日常工作中您每天都要使用 supplier 表许多次，而且您的英文打字的水平并不高。在这种情形下，就可以借助同义词来帮助您提高工作效率。

15.10　如何创建同义词

现在您就可以使用例 15-38 的 CREATE SYNONYM 语句为表 supplier 创建一个同义词（别名）s。

例 15-38

```
SQL> CREATE SYNONYM s
   2  FOR   supplier;
```

例 15-38 结果

```
同义词已创建。
```

现在您就可以把同义词（别名）s 当成 supplier 来使用，您可以使用例 15-39 的查询语句来验证这一点。

例 15-39

```
SQL> SELECT *
   2  FROM s;
```

例 15-39 结果

```
   S_CODE  SNAME                          CONTACT          PHONE            FAX
---------- ----------------- ---------------- ---------------- --------
      2000  仙客来百货                     张根发           4444944          4444844
      2010  心太软小商品                   石铁心           1741741          1741742
      2021  食为天餐具                     金元宝           1671671          1671674
      2032  食为先餐具                     陆合彩           1681684          1681684
```

那您可能会问，怎样才能知道到底拥有哪些同义词呢？

还记得数据字典 user_objects 吗？既然同义词是对象，它们在这个数据字典中就一定有记载。因此您可以使用例 15-41 的查询语句从数据字典 user_objects 中得到您所拥有的全部同义词的信息。当然，为了使 SQL*Plus 的显示输出更加清晰，您应该先使用例 15-40 的 SQL*Plus 格式化命令。

例 15-40

```
SQL> col object_name for a20
```

例 15-41

```
SQL> SELECT object_name, object_type, created, status
  2  FROM user_objects
  3  WHERE object_type LIKE 'SYN%';
```

例 15-41 结果

```
OBJECT_NAME          OBJECT_TYPE          CREATED    STATUS
-------------------- ------------------ ---------- ------
S                    SYNONYM              28-4月 -03 VALID
```

例 15-41 显示的结果表明，在您的模式（用户）下只有一个同义词，那就是您刚建立的 s。

但是例 15-41 显示的结果并没有告诉您这个同义词到底是基于哪个表的及表的主人是谁。如果您有这样一个同义词时，这方面的信息也许就显得特别重要了。您可以使用例 15-43 的查

询语句从数据字典 user_synonyms 中得到这方面的信息。当然，为了使 SQL*Plus 的显示输出更加清晰，您应该先使用例 15-42 的 SQL*Plus 格式化命令。

例 15-42

```
SQL> col table_owner for a12
SQL> col table_name for a12
```

例 15-43

```
SQL> SELECT synonym_name, table_owner, table_name
   2  FROM user_synonyms;
```

例 15-43 结果

```
SYNONYM_NAME                   TABLE_OWNER  TABLE_NAME
------------------------------ ------------ ----------
S                              SCOTT         SUPPLIER
```

例 15-43 显示的结果表明，在您的模式（用户）下只有一个同义词，那就是您刚建立的 s。该同义词基于表 supplier，而这个表的主人是 SCOTT。

创建同义词的语句格式如下：

CREATE [PUBLIC] SYNONYM　同义词的名字

FOR　对象名;

其中各个参数的含义如下。

- PUBLIC：系统中所有的用户都可以访问所创建的同义词。
- 同义词的名字：所创建的同义词的名字。
- 对象名：创建的同义词所基于的对象名。

在创建同义词时要注意的事项：

- 所基于的对象不能包含在任何软件包中。
- 一个私有的同义词不能与任何该用户下的其他对象重名。

您刚创建的同义词 s 是一个私有的同义词，即只能在您的用户（SCOTT）下直接引用，如果其他用户引用它就必须冠以用户名（即 scott.s 的方式引用），这样做很不方便。如果您所建的表 supplier 是一个所有用户共享并经常使用的表，您应该怎样处理这一问题呢？

15.11　创建公用同义词

您可以为 supplier 表创建一个公用同义词（Synonym）。为了演示方便，您应该从当前的 SCOTT 用户切换到另一个用户下，如 SYSTEM。您可以使用例 15-44 的 SQL*Plus 命令来完成用户的切换。

例 15-44

```
SQL> CONNECT SYSTEM/MANAGER
```

例 15-44 结果

```
已连接。
```

现在如果使用例 15-45 的查询语句，是不会得到您想要的信息的。

例 15-45

```
SQL> SELECT *
  2  FROM s;
```

例 15-45 结果

```
FROM s
     *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

您如果再使用例 15-46 的查询语句，也同样不会得到您想要的信息。

例 15-46

```
SQL> SELECT *
  2  FROM supplier;
```

例 15-46 结果

```
FROM supplier
     *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

如果您在同义词 s 之前冠以用户名（scott.），就可以得到您所需要的信息，如例 15-47 的查询语句。

例 15-47

```
SQL> SELECT *
  2  FROM scott.s;
```

例 15-47 结果

```
  S_CODE SNAME                 CONTACT          PHONE            FAX
---------- -------------------- ---------------- ---------------- ------
      2000 仙客来百货            张根发           4444944          4444844
      2010 心太软小商品          石铁心           1741741          1741742
      2021 食为天餐具            金元宝           1671671          1671674
      2032 食为先餐具            陆合彩           1681684          1681684
```

由于其他用户在引用这一同义词 s 时必须冠以用户名（.），这样很不方便，而且也容易产生输入错误，所以您可以使用例 15-48 的 DDL 语句为 SCOTT 用户下的 supplier 建立一个公用的同义词 ss。

例 15-48

```
SQL> CREATE PUBLIC SYNONYM ss
   2  FOR scott.supplier;
```

例 15-48 结果

```
同义词已创建。
```

现在您就可以利用刚创建的公用的同义词 ss 来获得您所希望得到的信息，您可以使用例 15-49 的查询语句。

例 15-49

```
SQL> SELECT *
   2  FROM ss;
```

例 15-49 结果

```
   S_CODE  SNAME                            CONTACT        PHONE            FAX
---------- ---------------- --------------- --------------- -------
     2000  仙客来百货                        张根发          4444944          4444844
     2010  心太软小商品                      石铁心          1741741          1741742
     2021  食为天餐具                        金元宝          1671671          1671674
     2032  食为先餐具                        陆合彩          1681684          1681684
```

您也可能会问，其他用户是否也可以使用同样的方法访问同义词 ss，答案是肯定的。如果还有疑问的话，您可以试着使用例 15-50 的 SQL*Plus 命令以 SYS 用户登录进入 Oracle 数据库。

例 15-50

```
SQL> CONNECT SYS/Oracle AS SYSDBA;
```

例 15-50 结果

```
已连接。
```

现在就可以利用公共同义词 ss 来获得您所希望得到的信息，您可以使用例 15-51 的查询语句。

例 15-51

```
SQL> SELECT *
   2  FROM ss;
```

例 15-51 结果

```
   S_CODE  SNAME                            CONTACT        PHONE            FAX
---------- --------------- --------------- --------------- -------
     2000  仙客来百货                        张根发          4444944          4444844
     2010  心太软小商品                      石铁心          1741741          1741742
     2021  食为天餐具                        金元宝          1671671          1671674
     2032  食为先餐具                        陆合彩          1681684          1681684
```

Oracle 并没有提供修改同义词的命令。如果您要修改某一同义词，您要先把它删除，然后再重新建立这个同义词。

15.12 删除同义词

如果经过了一段时间，您发现同义词 s 已经没什么用处了，您就可以使用例 15-53 的 DDL 语句将它删除。不过在这之前您可能要先使用例 15-52 的 SQL*Plus 命令重新注册到 SCOTT 用户下。

例 15-52

```
SQL> CONNECT SCOTT/TIGER
```

例 15-52 结果

```
已连接。
```

例 15-53

```
SQL> DROP SYNONYM s;
```

例 15-53 结果

```
同义词已丢弃。
```

虽然例 15-53 结果已经显示“同义词已丢弃”，但是为了谨慎起见，您还是应该使用例 15-54 的查询语句查看一下数据字典 user_objects。

例 15-54

```
SQL> SELECT object_name, object_type, created, status
  2  FROM user_objects
  3  WHERE object_type LIKE 'SYN%';
```

例 15-54 结果

```
未选定行。
```

或者您应该使用例 15-55 的查询语句查看一下数据字典 user_ synonyms。

例 15-55

```
SQL> SELECT synonym_name, table_owner, table_name
  2  FROM user_synonyms;
```

例 15-55 结果

```
未选定行。
```

例 15-54 和例 15-55 显示的结果进一步证实了您已经将同义词 s 成功地从系统中删除了。

删除同义词语句的格式如下：

DROP SYNONYM 同义词的名字;

15.13　应该掌握的内容

在学习下一章之前，请检查一下您是否已经掌握了以下的内容：

- 引入序列号的原因。
- 如何创建序列号？
- 序列号的特点。
- CREATE SEQUENCE 语句中的选项。
- 如何从数据字典中获得序列号的信息？
- 序列号的两个虚（伪）列 NEXTVAL 和 CURRVAL。
- 如何利用序列号向表中插入数据？
- 实际使用序列号时应该注意的问题。
- NEXTVAL 和 CURRVAL 的特点。
- 什么情况下可以使用 NEXTVAL 和 CURRVAL？
- 什么情况下不能使用 NEXTVAL 和 CURRVAL？
- 什么情况下可能造成序列号码的间隔？
- 如何修改序列号？
- 修改序列号时要注意的事项。
- 序列号的删除。
- 同义词引入的原因。
- 如何创建私有同义词？
- 如何创建公用同义词？
- 创建同义词时要注意的事项。
- 公用同义词的用法。
- 如何从数据字典中获得同义词的信息？
- 同义词的删除。

第16章

用户管理

在大型数据库环境中，可能有成百上千的用户在上面操作。在众多的用户当中难免有几位不守规矩的“武林高手”，特别是在互联网（Internet）环境中。这就使得数据库的安全变得极为重要。我们在这一章中将对 Oracle 数据库的安全及控制进行比较简单的介绍。

注意：

如果读者对这方面的内容非常感兴趣，或实际工作中要用到比较复杂的数据库的安全机制，可参阅 Oracle 体系结构（数据库管理）方面的书中的有关章节。

16.1　控制用户对数据库的访问

由于 Oracle 数据库管理系统的主要客户是大中型企业或机构，Oracle 数据库管理系统提供了一套相当完善的安全及控制机制（至少 Oracle 公司自己这样认为）。利用这套机制，您可以对 Oracle 数据库进行有效的安全管理。

其中包括：

- 利用口令或其他的安全机制来控制用户对数据库的访问。
- 利用系统权限限制用户对数据库中的系统资源的访问。
- 有效的系统权限管理。
- 利用对象权限限制用户对数据库中对象的访问。
- 利用视图限制用户对数据库中对象的访问。
- 为数据库中对象创建对应的同义词等。

Oracle 数据库的安全包括两大类：

- Oracle 数据库的系统安全。主要包括在系统一级来控制用户对数据库中的系统资源的访问和使用，如用户名和口令，用户可进行的系统操作和分配给用户的磁盘空间等。
- Oracle 数据库中数据的安全。主要包括控制用户对数据库中对象的访问和使用，以及控制用户对数据库中对象的操作等。

16.2　创建用户及给用户赋口令

Oracle 数据库管理系统构筑的第一道安全防线就是只允许合法的用户进入 Oracle 数据

库系统，这与实际生活中的许多例子非常近似，如许多国家在接受移民申请时要求申请人出示无犯罪记录证明，进入某一机构的工作人员必须持有有效的通行证等。

那么 Oracle 数据库管理系统是如何处理的呢？如果您想使用某一个 Oracle 数据库，则这个数据库的管理员（也可能是您本人）首先要为您在 Oracle 数据库系统上创建一个用户，并将一个密码（口令）赋予这个用户。

以后当您再次登录系统时，您就必须输入这个用户名和相应的密码（口令），Oracle 数据库管理系统会把您的输入与系统中所存的用户名和相应的密码（口令）进行比较，如果它们之间有任何不同的话，系统就会拒绝提供服务。

只有 DBA（用户）可以使用创建用户（CREATE USER）语句来创建一个用户。CREATE USER 语句的基本格式如下：

CREATE USER　用户名
IDENTIFIED BY　口令;

下面您就可以使用这一 DDL 语句创建一个名为 dog（狗）的用户，该用户的口令（密码）为 wangwang（汪汪）。如果您是以 SCOTT/TIGER 登录的，您使用例 16-1 的 CREATE USER 语句系统会报错的，因为 SCOTT 用户不具有 DBA（数据库管理员）的权限。

例 16-1

```
SQL> CREATE USER dog
  2  IDENTIFIED BY wangwang;
```

例 16-1 结果

```
IDENTIFIED BY wangwang
              *
ERROR 位于第 2 行:
ORA-01031: 权限不足
```

现在您可以使用例 16-2 的 SQL*Plus 命令切换到 DBA（数据库管理员）用户下，如 SYSTEM/MANAGER。

例 16-2

```
SQL> CONNECT SYSTEM/MANAGER
```

例 16-2 结果

```
已连接。
```

然后，您再执行与例 16-1 完全相同的 DDL 语句，如例 16-3。

例 16-3

```
SQL> CREATE USER dog
  2  IDENTIFIED BY wangwang;
```

例 16-3 结果

```
用户已创建。
```

这次系统的显示为“用户已创建”，这说明您已经成功地在系统中创建了 dog 这个用户。如果您想再次确认一下 dog 这个用户是否真的存在于系统中，您该怎么办呢？

您可以使用数据字典 dba_users，可以使用例 16-4 的查询语句来得到这方面的信息。

例 16-4

```
SQL> SELECT username, created
   2  FROM dba_users
   3  WHERE ROUND(created, 'DAY') = ROUND(SYSDATE, 'DAY');
```

例 16-4 结果

```
USERNAME                       CREATED
------------------------------ ------------
DOG                            29-4 月 -03
```

例 16-4 显示结果给出了 dog 用户的用户名和在系统中创建的时间。

您能说出例 16-4 中 WHERE 子句所用的 ROUND 函数的作用吗？它的作用是限制该查询语句的输出结果仅为那些当前建立的用户。如果您省略了这条 WHERE 子句，则这个查询语句的输出结果为系统中所有的用户。

既然您已经建立了 dog 用户，您可以试着利用这一用户登录系统。为了显示清楚，您可以在 SQL*Plus 中使用例 16-5 的 SQL*Plus 命令。

例 16-5

```
SQL> connect dog/wangwang;
```

例 16-5 结果

```
ERROR:
ORA-01045: user DOG lacks CREATE SESSION privilege; logon denied

警告: 您不再连接到 Oracle。
```

看到例 16-5 显示结果您可能会感到震惊，您仔细地检查了您在例 16-5 中输入的命令，但没有发现任何错误。这是为什么？

这时您开始认真地阅读例 16-5 显示结果中的错误提示信息。ORA-01045 的英文表明 DOG 用户没有 CREATE SESSION 这个系统权限，所以您无法登录。

当 DBA 使用 CREATE USER 语句成功地创建了一个用户后，此时该用户没有任何权限（就像人刚出生来到这个世界时也是两手空空什么也没有一样），接下来 DBA 就要授予这个用户相应的权限。

16.3 Oracle数据库管理系统中的权限

Oracle 数据库管理系统主要是通过权限来进行有效的安全管理和控制的。那么什么是权限呢？

权限是用来执行某些特定 SQL 语句的权力（能力）。权限又分为系统权限和对象权限。

- 系统权限是访问（使用）数据库（系统资源）的权力（能力）。
- 对象权限是维护数据库中的对象的权力（能力）。

数据库管理员（DBA）是数据库系统中最高级别的用户。DBA 具有数据库系统中的一切系统权限并拥有所有的系统资源。DBA 可以把这些权限的一些或全部授予其他的用户，也可以把这些系统资源的使用权授予其他的用户（如果他愿意的话）。

在这里需要介绍的另一个概念就是模式。那么什么是模式呢？

模式是一组对象的集合，如表、视图和序列号等。模式由数据库的用户所拥有，并且与用户具有相同的名字。其实，在实际使用中模式和用户是一回事。

在 Oracle 8 中有 80 多种系统权限，而在 Oracle 8i 中增加到 120 多种。详细讨论每一种系统权限的用法已经超出了本书的范围，本书只对如何使用系统权限作一简单的介绍。

以下是几个常用的系统权限。

- CREATE USER：创建其他的用户（需要具有 DBA 角色的权限）。
- DROP USER：删除其他的用户。
- SELECT ANY TABLE：查询任何用户的表和视图的权力。
- CREATE ANY TABLE：在任何模式中创建表。
- DROP ANY TABLE：删除任何模式中所创建的表。
- CREATE SESSION：连接数据库。
- CREATE TABLE：在用户自己的模式中创建表。
- CREATE VIEW：在用户自己的模式中创建视图。
- CREATE SEQUENCE：在用户自己的模式中创建序列号。
- CREATE PROCEDURE：在用户自己的模式中创建过程。

注意：

ANY 关键字表示在任何模式中都有定义的权限。在 Oracle 8 之前的版本中，具有 SELECT ANY TABLE 权限的用户可以查询数据字典。这可能存在安全隐患，因为为了开发软件的方便，一般高级开发人员常常被授予 SELECT ANY TABLE 权限。这样如果有高级开发人员想破坏系统，他们就能很容易地通过数据字典得到他们所需要的系统配置信息。在 Oracle 8 到 Oracle 8i 之间的版本中默认是指具有 SELECT ANY TABLE 权限的用户可以查询数据字典。但是 DBA 可以修改这一约定，使具有 SELECT ANY TABLE 权限的用户不能查询数据字典，在系统中只有 DBA 可以查询数据字典，这样系统会更安全。在 Oracle 9i 和以后的版本中默认是指具有 SELECT ANY TABLE 权限的用户不能查询数据字典，但是 DBA 可以把它的功能修改回 Oracle 8 之前的版本的约定。

16.4　如何将系统权限授予用户

介绍完了 Oracle 数据库管理系统中的权限，现在您就可以通过数据控制语言（Data

Control Language，DCL）中的 GRANT 语句为用户 dog 授予系统权限。在授权之前您应该先使用例 16-6 的 SQL*Plus 命令切换到 SYSTEM 用户下。

例 16-6

```
SQL> connect system/manager;
```

例 16-6 结果

```
已连接。
```

现在您就可以使用例 16-7 的 DCL 语句将 CREATE SESSION 这一系统权限授予 dog 用户。

例 16-7

```
SQL> GRANT CREATE SESSION TO dog;
```

例 16-7 结果

```
授权成功。
```

然后，您就可以再次使用例 16-8 的 SQL*Plus 的连接命令试着以 dog 用户登录。

例 16-8

```
SQL> connect dog/wangwang;
```

例 16-8 结果

```
已连接。
```

看到了例 16-8 显示的结果您知道终于如愿以偿了，您终于可以使用您所创建的用户登录 Oracle 系统了。此时您很想知道作为一个新用户，dog 到底有多少“家当”（即对象），可以使用例 16-9 的查询语句来获得这方面的信息。

例 16-9

```
SQL> SELECT object_name, created
   2  FROM user_objects;
```

例 16-9 结果

```
未选定行。
```

例 16-9 的结果告诉您，dog 一无所有，而且在系统中他什么也都不能建，因为 dog 没有相应的系统权限。您如果想让他干活就必须先赋予他应有的权限（所谓要想取之必先予之）。因此，您又一次使用例 16-10 的 SQL*Plus 命令切换到 SYSTEM 用户下。

例 16-10

```
SQL> connect system/manager;
```

例 16-10 结果

```
已连接。
```

为了进一步演示 GRANT 语句的用法，您可以使用例 16-11 的 CREATE USER 语句再

创建一个用户 cat（猫），该用户的密码为 miaomiao（喵喵）。

例 16-11

```
SQL> CREATE USER cat
  2  IDENTIFIED BY miaomiao;
```

例 16-11 结果

```
用户已创建。
```

然后，您先使用例 16-12 的 DCL 语句将 CREATE SESSION 这一系统权限授予 cat 用户。

例 16-12

```
SQL> GRANT CREATE SESSION
  2  TO cat;
```

例 16-12 结果

```
授权成功。
```

接下来您就可以使用例 16-13 的 DCL 语句将 SELECT ANY TABLE、CREATE TABLE、CREATE VIEW 这 3 个系统权限分别授予 dog 和 cat 用户。

例 16-13

```
SQL> GRANT SELECT ANY TABLE, CREATE TABLE, CREATE VIEW
  2  TO dog, cat;
```

例 16-13 结果

```
授权成功。
```

现在用户 dog 和 cat 具有完全相同的系统权限。我们以下的演示将使用新用户 cat，当然您也可以使用 dog 用户。于是您又一次使用例 16-14 的 SQL*Plus 命令，这次是切换到 cat 用户下。

例 16-14

```
SQL> connect cat/miaomiao;
```

例 16-14 结果

```
已连接。
```

因为现在 cat 用户有 SELECT ANY TABLE 的系统权限，所以他可以查看其他用户中的对象的信息。您可以使用例 16-15 的查询语句来查看 SCOTT 用户的 dept 表中的信息。

例 16-15

```
SQL> select * from scott.dept;
```

例 16-15 结果

```
   DEPTNO DNAME          LOC
---------- -------------- ---------
       10 ACCOUNTING     NEW YORK
```

```
        20  RESEARCH            DALLAS
        30  SALES               CHICAGO
        40  OPERATIONS          BOSTON
```

既然 cat 用户既有 SELECT ANY TABLE 又有 CREATE TABLE 的系统权限，因此您可以试着使用例 16-16 的 DDL 语句来创建一个新表 baby_cat（小猫）。

例 16-16

```
SQL> CREATE TABLE baby_cat
   2  AS
   3  SELECT *
   4  FROM scott.emp;
```

例 16-16 结果

```
FROM scott.emp
         *
ERROR 位于第 4 行:
ORA-01950: 表空间'SYSTEM'中无权限
```

如果您真的看到了如例 16-16 结果那样的错误信息，请不用担心，因为您没有做错任何事情（以下的内容您用不着去理解，因为这部分的内容是属于 Oracle 体系结构的。之所以在这里提一下是因为您在实际工作中可能会遇到这类问题，您只需按照命令输入系统即可）。

首先您必须使用例 16-17 的 SQL*Plus 命令，这次是切换到 SYSTEM（或其他 DBA）用户下。

例 16-17

```
SQL> connect system/manager;
```

例 16-17 结果

```
已连接。
```

然后，您可以使用例 16-18 的查询语句来查看在您的数据库中有几个表空间（tablespace）。

例 16-18

```
SQL> select tablespace_name from dba_tablespaces;
```

例 16-18 结果

```
TABLESPACE_NAME
---------------
SYSTEM
UNDOTBS
CWMLITE
DRSYS
EXAMPLE
INDX
TEMP
```

```
TOOLS
USERS
已选择 9 行。
```

在您的系统中有可能显示的结果有点差别。

现在您可以使用例 16-19 的 DDL 语句将 cat 用户的默认表空间改为 USERS（即当用户 cat 在创建表时如果没有指定表空间，系统就将该表建在 USERS 表空间上），并且 cat 用户最多可以在 USERS 表空间上使用 20MB 的磁盘空间。

例 16-19

```
SQL> ALTER USER cat
  2  DEFAULT TABLESPACE USERS
  3  QUOTA 20m ON USERS;
```

例 16-19 结果

```
用户已更改。
```

下面的内容您就应该理解了。您又一次使用例 16-20 的 SQL*Plus 命令切换到 cat 用户下。

例 16-20

```
SQL> connect cat/miaomiao;
```

例 16-20 结果

```
已连接。
```

现在您可以试着再使用例 16-21 的 DDL 语句来创建那个叫 baby_cat（小猫）的表。

例 16-21

```
SQL>  CREATE TABLE baby_cat
  2   AS
  3   SELECT *
  4   FROM scott.emp;
```

例 16-21 结果

```
表已创建。
```

看到了例 16-21 显示的结果，您终于感到胜利离您只有一条 SQL 语句之遥了。接下来可以使用例 16-22 的查询语句来查看您刚创建的表的内容。

例 16-22

```
SQL> SELECT ename
  2  FROM baby_cat;
```

例 16-22 结果

```
ENAME
-------------
SMITH
```

```
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER
已选择 14 行。
```

16.5 如何查看用户具有的系统权限

通过创建 baby_cat 表的艰辛历程，您可确定用户 cat（猫）至少拥有 CREATE SESSION、SELECT ANY TABLE 和 CREATE TABLE 系统权限。那么有没有一种办法能准确地知道一个用户究竟拥有多少系统权限呢？

答案是有。您可以使用数据字典 SESSION_PRIVS 来得到一个用户所拥有的全部系统权限的信息。例如，您可以使用例 16-23 的查询语句来查看用户 cat 到底拥有多少系统权限。

例 16-23

```
SQL> SELECT *
  2  FROM SESSION_PRIVS;
```

例 16-23 结果

```
PRIVILEGE
----------------
CREATE SESSION
CREATE TABLE
SELECT ANY TABLE
CREATE VIEW
```

例 16-23 显示的结果表明，用户 cat 只拥有 4 个您用 GRANT 语句授予的系统权限，一个不多，一个也不少。

检查完了 cat 用户，您也想检查 dog 用户所拥有的系统权限，所以应该先使用例 16-24 的 SQL*Plus 命令切换到 dog 用户。

例 16-24

```
SQL> CONNECT dog/wangwang;
```

例 16-24 结果

```
已连接。
```

然后，您就可以使用例 16-25 的查询语句来查看用户 dog 到底拥有多少系统权限。

例 16-25

```
SQL> SELECT *
   2  FROM SESSION_PRIVS;
```

例 16-25 结果

```
PRIVILEGE
----------------
CREATE SESSION
CREATE TABLE
SELECT ANY TABLE
CREATE VIEW
```

果然没逃出您的神机妙算，例 16-25 显示的结果表明，用户 dog 也只拥有 4 个您用 GRANT 语句授予的系统权限，这 4 个系统权限与 cat 用户所拥有的一模一样。

从开始学习这本书起，您就开始使用 SCOTT 这一用户，但是您一直不清楚 SCOTT 用户到底拥有多少系统权限（没关系，所谓“难得糊涂”嘛）。现在也许想清醒一下，想知道 SCOTT 用户到底拥有多少系统权限。但是您必须先使用例 16-26 的 SQL*Plus 命令切换到 SCOTT 用户。

例 16-26

```
SQL> CONNECT scott/tiger
```

例 16-26 结果

```
已连接。
```

然后，您就可以使用例 16-27 的查询语句来查看 SCOTT 用户到底拥有多少系统权限。

例 16-27

```
SQL> SELECT *
   2  FROM SESSION_PRIVS;
```

例 16-27 结果

```
PRIVILEGE
--------------------
CREATE SESSION
ALTER SESSION
UNLIMITED TABLESPACE
CREATE TABLE
CREATE CLUSTER
CREATE SYNONYM
```

```
CREATE VIEW
CREATE SEQUENCE
CREATE DATABASE LINK
CREATE PROCEDURE
CREATE TRIGGER
CREATE TYPE
CREATE OPERATOR
CREATE INDEXTYPE
已选择 14 行。
```

看了例 16-27 显示的结果您一定吃惊不小，SCOTT 用户有这么多的系统权限，这也是为什么您利用 SCOTT 用户几乎做完了本书全部例题的原因所在。

16.6 引入角色的原因

相信通过前面几节的学习，您已经掌握了如何将系统权限授予用户，但是这种方法存在着缺陷。

假设您现在正在管理一个有 1000 个用户的大型系统，而且这些用户是陆续建立的，每个用户需要 20 个系统权限。为了简化问题，我们进一步假设所有的用户都需要完全一样的系统权限。

现在您如果使用前面几节学过的方法将这 20 种系统权限分别授予这 1000 个用户，所有的系统权限使用的次数将达到 20×1000=20000 次。另外，如果用户使用的系统权限需要修改时，其修改量也是惊人的。

为了解决这一难题，Oracle 引入了角色（Role）。那么什么是角色呢？

角色是一组命名的系统权限，这组系统权限可以通过该名字授予用户。

通过使用角色会使您的授予和回收系统权限的维护工作简单得多。如上面的例子，现在您就可以通过角色来对用户授权。您先把 20 种系统权限都授予角色，然后您再把这个角色分别授予这 1000 个用户。这样，所有的系统权限使用的次数就将变为 20+1000 = 1020 次。看看省了您多少工作，真是苦干不如巧干。通过使用角色，在用户使用的系统权限需要修改时，其修改量明显下降。想想看，这是为什么？

使用角色的另一个好处是可以提高系统的效率。听起来是不是有点太神了（有点像包治百病的祖传密方了）。下面我们就来解释其中的玄机。

无论您是使用直接授权还是通过角色授权，最终有关用户权限的信息都要记录到数据字典中（磁盘上）。还是继续使用前面的例子。如果您使用的是直接授权法，记录到数据字典中（磁盘上）的有关用户权限的信息为 20000 项。而当使用角色授权的方法时，记录到数据字典中（磁盘上）的有关用户权限的信息就只有 1020 项。磁盘上的数据少了，查询时的 I/O 也就少了，因此 Oracle 服务器查询数据字典的速度也就会加快很多，从而提高了系统的效率。

注意：

以上解释只是为了帮助读者理解，其实这样的解释是相当不正规的。

16.7　角色的创建和使用

为了演示角色（Role）的创建和使用，我们首先收回用户 dog 和 cat 的所有权限。您应该做的第一件事是切换到 SYSTEM 用户，因为只有 DBA 用户才有权使用这一节的大多数命令。您可以使用例 16-28 的 SQL*Plus 命令切换到 SYSTEM 用户。

例 16-28

```
SQL> connect system/manager
```

例 16-28 结果

```
已连接。
```

然后，您就可以使用例 16-29 的 DCL 语句收回用户 dog 和 cat 的所有权限（其中的 REVOKE 语句以后会详细介绍）。

例 16-29

```
SQL> REVOKE SELECT ANY TABLE, CREATE TABLE,
  2         CREATE VIEW, CREATE SESSION
  3  FROM dog, cat;
```

例 16-29 结果

```
撤销成功。
```

虽然例 16-29 的结果显示“撤销成功。”，但为了慎重起见，您还是应该使用例 16-30 和例 16-31 的 SQL*Plus 命令来验证一下。

例 16-30

```
SQL> connect dog/wangwang
```

例 16-30 结果

```
ERROR:
ORA-01045: user DOG lacks CREATE SESSION privilege; logon denied

警告: 您不再连接到 Oracle。
```

例 16-31

```
SQL> connect cat/miaomiao
```

例 16-31 结果

```
ERROR:
ORA-01045: user CAT lacks CREATE SESSION privilege; logon denied
```

现在您应该使用例 16-32 的 SQL*Plus 命令重新连接到 SYSTEM 用户。

例 16-32

```
SQL> connect system/manager
```

例 16-32 结果

已连接。

为了更好地演示角色（Role）的创建和使用，我们再创建一个用户 pig（猪），该用户的密码为 hengheng。您可以使用例 16-33 的 DDL 语句来创建用户 pig（猪）。

例 16-33

```
SQL> CREATE USER pig
   2  IDENTIFIED BY hengheng;
```

例 16-33 结果

用户已创建。

现在您就可以使用例 16-34 的 DDL 语句来创建角色 animal（动物）。

例 16-34

```
SQL> CREATE ROLE animal;
```

例 16-34 结果

角色已创建。

当成功地创建了角色 animal 之后，您就可以使用例 16-35 的 DCL 语句将系统权限 SELECT ANY TABLE、CREATE TABLE、CREATE VIEW 和 CREATE SESSION 授予角色 animal。

例 16-35

```
SQL> GRANT SELECT ANY TABLE, CREATE TABLE,
   2        CREATE VIEW, CREATE SESSION
   3  TO animal;
```

例 16-35 结果

授权成功。

最后，您可以使用例 16-36 的 DCL 语句将角色 animal 分别授予用户 dog、cat 和 pig。

例 16-36

```
SQL> GRANT animal TO dog, cat, pig;
```

例 16-36 结果

授权成功。

假设当您将角色 animal 分别授予用户 dog、cat 和 pig 之后，您就休假去了。当您一个多月后回到公司时，您又想使用角色 animal。但此时除了角色名您还有点模糊的印象其他的什么都记不得了（可能是假期太愉快了）。此时，您就可以使用例 16-37 的查询语句通

过数据字典 ROLE_SYS_PRIVS 来得到角色 animal 所拥有的系统权限。

例 16-37

```
SQL> SELECT *
    2  FROM ROLE_SYS_PRIVS
    3  WHERE ROLE LIKE 'ANI%';
```

例 16-37 结果

```
ROLE                       PRIVILEGE                                  ADM
---------------  ----------------------------------------  --------
ANIMAL                     CREATE SESSION                             NO
ANIMAL                     CREATE TABLE                               NO
ANIMAL                     CREATE VIEW                                NO
ANIMAL                     SELECT ANY TABLE                           NO
```

您也可以使用例 16-38 的查询语句通过数据字典 dba_role_privs 来得到所有授予 animal 角色的用户。

例 16-38

```
SQL> SELECT *
    2  FROM dba_role_privs
    3  WHERE GRANTED_ROLE LIKE 'ANI%';
```

例 16-38 结果

```
GRANTEE                    GRANTED_ROLE                 ADM  DEF
----------------------------  -------------------------  ---  ---
CAT                       ANIMAL                        NO   YES
DOG                       ANIMAL                        NO   YES
PIG                       ANIMAL                        NO   YES
SYSTEM                    ANIMAL                        YES  YES
```

注意：

以上两个数据字典要在 DBA（如 SYSTEM）用户中使用。

您也可以通过使用例 16-39~例 16-44 的命令来分别得到用户 dog、cat 和 pig 所拥有的系统权限。

例 16-39

```
SQL> connect dog/wangwang
```

例 16-39 结果

```
已连接。
```

例 16-40

```
SQL> SELECT *
   2  FROM SESSION_PRIVS;
```

例 16-40 结果

```
PRIVILEGE
----------------
CREATE SESSION
CREATE TABLE
SELECT ANY TABLE
CREATE VIEW
```

例 16-41

```
SQL> connect cat/miaomiao
```

例 16-41 结果

```
已连接。
```

例 16-42

```
SQL> SELECT *
  2  FROM SESSION_PRIVS;
```

例 16-42 结果

```
PRIVILEGE
----------------
CREATE SESSION
CREATE TABLE
SELECT ANY TABLE
CREATE VIEW
```

例 16-43

```
SQL> connect pig/hengheng
```

例 16-43 结果

```
已连接。
```

例 16-44

```
SQL> SELECT *
  2  FROM SESSION_PRIVS;
```

例 16-44 结果

```
PRIVILEGE
----------------
CREATE SESSION
CREATE TABLE
SELECT ANY TABLE
CREATE VIEW
```

介绍完了系统权限和角色，下面将讨论对象的权限。

16.8　对象的权限和授权语句

Oracle 系统中共有 8 种对象的权限（Object Privilege），它们分别是 EXECUTE、ALTER、SELECT、INDEX、REFERENCES、INSERT、UPDATE 和 DELETE。表 16.1 可能会对您理解对象的权限和对象之间的关系有所帮助。

表　16.1

Object Privilege	Procedure	Sequence	View	Table
EXECUTE	Y			
ALTER		Y		Y
SELECT		Y	Y	Y
INSERT			Y	Y
UPDATE			Y	Y
DELETE			Y	Y
INDEX				Y
REFERENCES				Y

参照表 16.1，我们给出以下简单的解释以帮助您理解和记忆对象的权限和对象之间的关系。

- 过程（Procedure）：只能执行（EXECUTE），因为过程是程序代码，它里面不能存数据，所以有关数据的操作对它都不适用。这里 Procedure 包含了函数（Function）和软件包（Package）。
- 序列号（Sequence）：它不是程序代码，所以不能执行（EXECUTE）。它里面也不能存数据，所以有关数据的操作对它基本都不适用。但它有两个伪列 NEXTVAL 和 CURRVAL，因此您可以对它进行 SELECT 操作和 ALTER 操作。
- 视图（View）：Oracle 规定不能对视图进行修改（ALTER）操作。如果要修改一个视图，应先将其删除，然后再重建。视图也不是程序代码，所以不能执行（EXECUTE）。真正的数据都存在所引用的表中。所有对视图数据的操作都由系统转换成对所引用的表中相应数据的操作。所以在视图上不能进行索引（INDEX）操作和引用（REFERENCES）操作，即创建外键。因此，在视图上只能进行 SELECT 和 DML（INSERT、UPDATE 和 DELETE）操作。
- 表（Table）：它不是程序代码，所以不能执行（EXECUTE）。它是数据库的最基本的对象，所有数据都存在表中，因此，所有有关数据的操作对它都适用。

对象的主人（拥有者）自动具有该对象的一切“生杀大权”（对象的权限）。对象的拥有者可以把一些对象的权限授予其他用户。

将对象的权限授予其他用户的 GRANT 语句的格式如下：

GRANT　对象的权限 |ALL[(列名[, 列名…])]

ON　　对象名

TO　　[用户名|角色名|PUBLIC]

[WITH GRANT OPTION]

其中各参数的含义如下。

- 对象的权限：要授予的对象的权限。
- ALL：所有对象的权限。
- 列名：该列上的对象权限要授予其他的用户。
- ON 对象名：该对象上的对象权限要授予其他的用户。
- TO [用户名|角色名]：指明对象权限要授予谁[某个用户|某个角色]。
- PUBLIC：指明对象权限要授予所有的用户。
- WITH GRANT OPTION：允许被授予的用户再将这些对象权限授予其他用户。

16.9 对象权限授权实例

首先您可以使用例 16-45 的 SQL*Plus 命令连接到 pig 用户。

例 16-45

```
SQL> connect pig/hengheng;
```

例 16-45 结果

```
已连接。
```

现在您可以使用例 16-46 的查询语句查看 pig 用户到底有多少家当。

例 16-46

```
SQL> select * from cat;
```

例 16-46 结果

```
未选定行
```

例 16-46 显示的结果表明，他（pig）是一名地地道道的无产者，一无所有。这是因为自从您把他带到这个世界上（创建了 pig 用户）以来，您什么也没给（授予）他，他也什么都没创建过。

但是当您使用例 16-47 的查询语句时，您却可以得到 cat 用户的 baby_cat 中的内容，这是因为 pig 用户的系统权限是通过角色 animal 授予的，而角色 animal 中的系统权限包括了 SELECT ANY TABLE。

例 16-47

```
SQL> select * from cat.baby_cat;
```

例 16-47 结果

```
     EMPNO ENAME      JOB              MGR HIREDATE
---------- ---------- --------- ---------- ---------- ---
      7369 SMITH      CLERK           7902 17-12 月-1980
```

```
      7499 ALLEN      SALESMAN       7698 20-2 月 -1981
      7521 WARD       SALESMAN       7698 22-2 月 -1981
      7566 JONES      MANAGER        7839 02-4 月 -1981
      7654 MARTIN     SALESMAN       7698 28-9 月 -1981
      7698 BLAKE      MANAGER        7839 01-5 月 -1981
      7782 CLARK      MANAGER        7839 09-6 月 -1981
      7788 SCOTT      ANALYST        7566 19-4 月 -1987
      7839 KING       PRESIDENT           17-11 月-1981
      7844 TURNER     SALESMAN       7698 08-9 月 -1981
      7876 ADAMS      CLERK          7788 23-5 月 -1987
      7900 JAMES      CLERK          7698 03-12 月-1981
      7902 FORD       ANALYST        7566 03-12 月-1981
      7934 MILLER     CLERK          7782 23-1 月 -1982
已选择 14 行。
```

注意：

为了显示清楚，例 16-47 结果只给出了部分显示。

为了演示对象权限授予方便，我们应该首先从角色 animal 中收回系统权限 SELECT ANY TABLE。您应该做的第一件事是切换到 SYSTEM 用户，因为只有 DBA 用户才有权使用 SQL 的收回（REVOKE）语句。您可以使用例 16-48 的 SQL*Plus 命令切换到 SYSTEM 用户。

例 16-48

```
SQL> connect system/manager
```

例 16-48 结果

```
已连接。
```

然后，您就可以使用例 16-49 的 DCL 语句收回角色 animal 的系统权限 SELECT ANY TABLE。

例 16-49

```
SQL> REVOKE SELECT ANY TABLE
   2  FROM animal;
```

例 16-49 结果

```
撤销成功。
```

为了检验 pig 用户是否真的失去了 SELECT ANY TABLE 系统权限，您应该首先使用例 16-50 的 SQL*Plus 命令切换到 pig 用户。

例 16-50

```
SQL> connect pig/hengheng;
```

例 16-50 结果

```
已连接。
```

然后，您就可以使用例 16-51 的查询语句来验证 pig 用户是否真的失去了 SELECT ANY TABLE 系统权限。

例 16-51

```
SQL> SELECT ename
   2  FROM cat.baby_cat;
```

例 16-51 结果

```
FROM cat.baby_cat
        *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

从例 16-51 显示的结果判断，pig 用户似乎真的失去了 SELECT ANY TABLE 系统权限。但是为了保险起见，您还是应该使用例 16-52 的查询语句进一步确认。

例 16-52

```
SQL> SELECT *
  2   FROM SESSION_PRIVS;
```

例 16-52 结果

```
PRIVILEGE
--------------
CREATE SESSION
CREATE TABLE
CREATE VIEW
```

看了例 16-52 显示的结果，您心里终于踏实了。现在一切准备就绪，于是您使用例 16-53 的 SQL*Plus 命令切换到 cat 用户。

例 16-53

```
SQL> connect cat/miaomiao;
```

例 16-53 结果

```
已连接。
```

终于熬到了您为其他用户授权的时刻了。您使用例 16-54 的 DCL 语句将 cat 用户的 baby_cat 表的 SELECT（查询）权限授予 pig 用户。该语句中的 WITH GRANT OPTION 选项表示 pig 用户可以将这一对象的权限再授予其他的用户。

例 16-54

```
SQL> GRANT select
   2  ON    baby_cat
   3  TO    pig
   4  WITH GRANT OPTION;
```

例 16-54 结果

```
授权成功。
```

注意：

在 GRANT 语句中使用 WITH GRANT OPTION 选项要非常谨慎。因为使用不当的话，可能会造成对该对象安全控制的失控。例如，虽然 cat（猫）大哥可以对 pig（猪）小弟百分之百地放心（他们可能是出生入死的钢哥们），但 pig（猪）小弟的手下或奴才就很难说了。

然后，您可以开始测试 pig 用户是否有权访问 cat 用户的 baby_cat 表。您应该先使用例 16-55 的 SQL*Plus 命令切换到 pig 用户。

例 16-55

```
SQL> connect pig/hengheng;
```

例 16-55 结果

```
已连接。
```

现在您可以使用例 16-56 的查询语句来测试 pig 用户是否有权访问 cat 用户的 baby_cat 表。

例 16-56

```
SQL> SELECT empno, ename
  2  FROM cat.baby_cat;
```

例 16-56 结果

```
     EMPNO ENAME
---------- ------
      7369 SMITH
      7499 ALLEN
      7521 WARD
      7566 JONES
      7654 MARTIN
      7698 BLAKE
      7782 CLARK
      7788 SCOTT
      7839 KING
      7844 TURNER
      7876 ADAMS
      7900 JAMES
      7902 FORD
      7934 MILLER
已选择 14 行。
```

现在您可以开始测试 dog 用户是否有权访问 cat 用户的 baby_cat 表。您应该先使用例 16-57 的 SQL*Plus 命令切换到 dog 用户。

例 16-57

```
SQL> connect dog/wangwang;
```

例 16-57 结果

```
已连接。
```

您可以使用例 16-58 的查询语句来测试 dog 用户是否有权访问 cat 用户的 baby_cat 表。

例 16-58

```
SQL> SELECT empno, ename
  2  FROM cat.baby_cat;
```

例 16-58 结果

```
FROM cat.baby_cat
       *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

看到了例 16-58 显示的结果您意识到，还没有把 cat 用户中的 baby_cat 表的 SELECT（查询）权限授予 dog 用户。

于是您使用例 16-59 的 SQL*Plus 命令切换到 pig 用户。

例 16-59

```
SQL> connect pig/hengheng;
```

例 16-59 结果

```
已连接。
```

您可以使用例 16-60 的 DCL 语句将 cat 用户的 baby_cat 表的 SELECT（查询）权限授予 dog 用户。

例 16-60

```
SQL> GRANT select
  2  ON    cat.baby_cat
  3  TO    dog;
```

例 16-60 结果

```
授权成功。
```

看到例 16-60 的结果显示“授权成功。”时，pig（猪）可能在想：“cat（猫）大哥真仗义，自己当了官，有了好处都没忘了我 pig（猪）小弟。真得感谢上苍，这回跟对了主子，不但我自己有了权还可以把手中的权力分给我的狐朋狗友。”

然后，您应该先使用例 16-61 的 SQL*Plus 命令切换到 dog 用户。

例 16-61

```
SQL> connect dog/wangwang;
```

例 16-61 结果

```
已连接。
```

然后，您可以使用例 16-62 的查询语句来测试 dog 用户是否有权访问 cat 用户的 baby_cat 表。

例 16-62

```
SQL> SELECT empno, ename
  2  FROM cat.baby_cat;
```

例 16-62 结果

```
    EMPNO ENAME
---------- ------
     7369 SMITH
     7499 ALLEN
     7521 WARD
     7566 JONES
     7654 MARTIN
     7698 BLAKE
     7782 CLARK
     7788 SCOTT
     7839 KING
     7844 TURNER
     7876 ADAMS
     7900 JAMES
     7902 FORD
     7934 MILLER
已选择 14 行。
```

您还记得 SCOTT 用户中的 supplier 表吗？假设这个表是公司数据库中核心的表，必须所有数据库用户都能够访问它。现在您试着使用例 16-63 的查询语句来浏览 SCOTT 用户的 supplier 表中的内容。

例 16-63

```
SQL> SELECT *
  2  FROM scott.supplier;
```

例 16-63 结果

```
FROM scott.supplier
          *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

看到了例 16-63 显示的结果您马上意识到，您还没有把 SCOTT 用户中 supplier 表的 SELECT（查询）权限赋予 cat、dog 和 pig 3 个新创建的用户。于是，您马上使用例 16-64

的 SQL*Plus 命令切换到 SCOTT 用户。

例 16-64

```
SQL> connect scott/tiger;
```

例 16-64 结果

```
已连接。
```

现在您就可以使用例 16-5 的 DCL 语句将 SCOTT 用户中 supplier 表的 SELECT（查询）权限赋予所有的用户（看来 SCOTT 这小子真是活雷锋，自己费了九牛二虎的力气建的表居然就这么轻易地让大家共享，连一分钱也不收，太不可思议了）。

例 16-65

```
SQL> GRANT select
  2  ON    supplier
  3  TO    PUBLIC;
```

例 16-65 结果

```
授权成功。
```

然后，您可以先使用例 16-66 的 SQL*Plus 命令切换到 dog 用户。

例 16-66

```
SQL> connect dog/wangwang
```

例 16-66 结果

```
已连接。
```

此时，您可以使用例 16-67 的查询语句来测试 dog 用户是否有权访问 SCOTT 用户中的 supplier 表。

例 16-67

```
SQL> SELECT *
  2  FROM scott.supplier;
```

例 16-67 结果

```
    S_CODE  SNAME                          CONTACT          PHONE            FAX
---------- ---------------- ---------------- ---------------- -------
      2000  仙客来百货                      张根发           4444944          4444844
      2010  心太软小商品                    石铁心           1741741          1741742
      2021  食为天餐具                      金元宝           1671671          1671674
      2032  食为先餐具                      陆合彩           1681684          1681684
```

为了检验是不是所有的用户真的都可以访问 SCOTT 用户中的 supplier 表，您又使用了例 16-68 的 SQL*Plus 命令切换到 cat 用户。

例 16-68

```
SQL> connect cat/miaomiao
```

例 16-68 结果

```
已连接。
```

此时，您又可以使用例 16-69 的查询语句来测试 cat 用户是否有权访问 SCOTT 用户中 supplier 表。

例 16-69

```
SQL> SELECT *
  2  FROM scott.supplier;
```

例 16-69 结果

```
  S_CODE SNAME                            CONTACT          PHONE            FAX
---------- ---------------- ---------------- ---------------- -------
    2000 仙客来百货                          张根发          4444944          4444844
    2010 心太软小商品                        石铁心          1741741          1741742
    2021 食为天餐具                          金元宝          1671671          1671674
    2032 食为先餐具                          陆合彩          1681684          1681684
```

依此类推，您也可以使用相同的方法测试 pig 用户。

这样测试的结果是一定的，即所有的用户都可以访问 SCOTT 用户中的 supplier 表。现在您可能会问如果您在例 16-65 的 DCL 语句执行之后新创建了一个用户，这个用户能访问 SCOTT 用户中的 supplier 表吗？您可以通过以下的例子来自己找出答案。

您应该做的第一件事是切换到 SYSTEM 用户，因为只有 DBA 用户才有权创建新用户。您可以使用例 16-70 的 SQL*Plus 命令切换到 SYSTEM 用户。

例 16-70

```
SQL> connect system/manager
```

例 16-70 结果

```
已连接。
```

然后，您可以使用例 16-71 的 DDL 语句创建一个叫 fox（狐狸）的新用户，该用户的口令为 loveyou（爱您）。

例 16-71

```
SQL> CREATE USER fox
  2  IDENTIFIED BY loveyou;
```

例 16-71 结果

```
用户已创建。
```

现在您就可以使用例 16-72 的 DCL 语句将角色 animal 授予用户 fox。

例 16-72

```
SQL> GRANT animal TO fox;
```

例 16-72 结果

```
授权成功。
```

为了检验 fox 用户是否可以访问 SCOTT 用户中的 supplier 表，您使用了例 16-73 的 SQL*Plus 命令以切换到 fox 用户。

例 16-73

```
SQL> connect fox/loveyou;
```

例 16-73 结果

```
已连接。
```

此时，您又可以使用例 16-74 的查询语句来测试 fox 用户是否有权访问 SCOTT 用户中的 supplier 表。

例 16-74

```
SQL> SELECT *
   2  FROM scott.supplier;
```

例 16-74 结果

```
  S_CODE  SNAME                            CONTACT          PHONE            FAX
---------- ---------------- ---------------- ---------------- --------
      2000  仙客来百货                       张根发          4444944          4444844
      2010  心太软小商品                     石铁心          1741741          1741742
      2021  食为天餐具                       金元宝          1671671          1671674
      2032  食为先餐具                       陆合彩          1681684          1681684
```

看到例 16-74 显示的结果，您应该已经知道答案了。

以上检测对象权限都是通过查询语句间接得到的。那么有没有一种直接的方法呢？有，您可以通过查询数据字典 user_tab_privs_made 得到。

现在，您可以使用例 16-75 的 SQL*Plus 命令切换到 cat 用户。

例 16-75

```
SQL> connect cat/miaomiao;
```

例 16-75 结果

```
已连接。
```

为了使显示清晰，您应该使用例 16-76 的 SQL*Plus 命令来先格式化显示输出。

例 16-76

```
SQL> col GRANTEE for a8
SQL> col TABLE_NAME for a10
SQL> col GRANTOR for a8
SQL> col PRIVILEGE for a18
```

此时，您就可以使用例 16-77 的查询语句来查看在 cat 用户下的对象和对象权限之间的

关系。

例 16-77

```
SQL> SELECT *
   2 FROM user_tab_privs_made;
```

例 16-77 结果

```
GRANTEE  TABLE_NAME GRANTOR  PRIVILEGE         GRA HIE
-------- ---------- -------- ----------------- --- ---
PIG      BABY_CAT   CAT      SELECT            YES NO
DOG      BABY_CAT   PIG      SELECT            NO  NO
```

例 16-77 显示的结果的第 1 行表明：cat 用户把 baby_cat 的 SELECT 对象权限授予了 pig 用户，而且 pig 用户还可以将这一对象权限授予其他的用户。

例 16-77 显示的结果的第 2 行表明：pig 用户把 baby_cat 的 SELECT 对象权限授予了 dog 用户，但是 dog 用户不能将这一对象权限授予其他的用户。

您还可以只把某一对象的某些列的对象权限授予其他的用户。为了演示方便，您应该使用例 16-78 的 SQL*Plus 命令切换到 SCOTT 用户。

例 16-78

```
SQL> connect scott/tiger
```

然后，您可以使用例 16-79 的 DCL 语句将 SCOTT 用户的 supplier 表中 phone 和 fax 的修改权限（update）授予 cat 用户。

例 16-79

```
SQL> GRANT update(phone, fax)
   2 ON   supplier
   3 TO   cat;
```

例 16-79 结果

```
授权成功。
```

您可以使用例 16-80 的 SQL*Plus 命令切换到 cat 用户。

例 16-80

```
SQL> connect cat/miaomiao;
```

例 16-80 结果

```
已连接。
```

您现在就可以使用例 16-81 和例 16-82 的 DML 语句试着修改 SCOTT 用户的 supplier 表中 phone 和 fax 的数据。

例 16-81

```
SQL> UPDATE scott.supplier
   2 SET phone = '168 cat'
   3 WHERE s_code = 2000;
```

例 16-81 结果

```
已更新 1 行。
```

例 16-82

```
SQL> UPDATE scott.supplier
  2  SET fax = 'fox and dog'
  3  WHERE s_code = 2010;
```

例 16-82 结果

```
已更新 1 行。
```

最后，别忘了使用例 16-83 的查询语句检查您的修改是否真的成功了。

例 16-83

```
SQL> SELECT *
  2  FROM scott.supplier;
```

例 16-83 结果

S_CODE	SNAME	CONTACT	PHONE	FAX
2000	仙客来百货	张根发	168 cat	4444844
2010	心太软小商品	石铁心	1741741	fox and dog
2021	食为天餐具	金元宝	1671671	1671674
2032	食为先餐具	陆合彩	1681684	1681684

例 16-83 显示的结果表明，您已成功修改了 SCOTT 用户的 supplier 表中 phone 和 fax 的数据。

如果您细心地看一下例 16-83 显示的结果，您就会发现第 1 行中的电话（PHONE）和第 2 行中的传真（FAX）肯定是有问题的，因为现实中电话和传真的号码只能为数字。

您也可能想如果将这两列的数据类型改为数字型，问题不就解决了吗？的确这一问题是解决了，但是一个新的问题又出现了。因为数字型的数据是可以进行算术运算的，而电话和传真的号码是不能进行算术运算的。您此时就要考虑如何防止对电话和传真的号码进行算术运算。

在现实中，几乎每一种设计都有它的优点和缺陷。作为系统的设计者，您要根据实际情况进行取舍。您永远不要梦想设计一个完美的系统，因为您永远没有足够的资源（时间、金钱和人力等）。您的做法应该是先设计一个用户可以接受的系统，然后再进行完善。

16.10 权限的回收

您可以使用 REVOKE 语句来回收权限。

回收系统权限的语句格式如下：

REVOKE {系统权限|角色名} [, {系统权限|角色名}]…

FROM　{用户名|角色名| PUBLIC } [, {用户名|角色名|PUBLIC}]…

您已经在例 16-49 中利用这一语句收回了角色 animal 的系统权限 SELECT ANY TABLE。

回收对象权限的语句格式如下：

REVOKE {对象权限 [, 对象权限…]|ALL}

ON　对象名

FROM　{用户名|角色名| PUBLIC } [, {用户名|角色名|PUBLIC}]…

CASCAD CONSTRAINTS];

其中，CASCAD CONSTRAINTS 为处理引用完整性时所需要的。

为了收回 pig 用户授予 dog 用户的对象权限，您可使用例 16-84 的 SQL*Plus 命令切换到 pig 用户。

例 16-84

```
SQL> connect pig/hengheng;
```

例 16-84 结果

```
已连接。
```

现在您可以使用例 16-85 的 DCL 语句将 cat 用户的 baby_cat 表的查询（SELECT）权限从 dog 用户收回。

例 16-85

```
SQL> REVOKE select
  2  ON     cat.baby_cat
  3  FROM   dog;
```

例 16-85 结果

```
撤销成功。
```

为了使演示更加清楚，您可以使用例 16-86 带有 WITH GRANT OPTION 选项的 DCL 语句将 cat 用户的 baby_cat 表的查询（SELECT）权限再次授予 dog 用户。

例 16-86

```
SQL> GRANT select
  2  ON     cat.baby_cat
  3  TO     dog
  4  WITH GRANT OPTION;
```

例 16-86 结果

```
授权成功。
```

然后，您应该使用例 16-87 的 SQL*Plus 命令切换到 dog 用户。

例 16-87

```
SQL> connect dog/wangwang;
```

例 16-87 结果

```
已连接。
```

别忘了使用例 16-88 的查询语句检查您所做的授权是否真的成功了。

例 16-88

```
SQL> SELECT rownum "Cat NO.", ename "Cat Name"
  2  FROM  (SELECT ename
  3         FROM cat.baby_cat
  4         ORDER BY ename)
  5  WHERE ROWNUM <= 6;
```

例 16-88 结果

```
  Cat NO. Cat Name
---------- --------
         1 ADAMS
         2 ALLEN
         3 BLAKE
         4 CLARK
         5 FORD
         6 JAMES
已选择 6 行。
```

还记得例 16-88 的查询语句吗？这是一个前 n（6）行的查询（Top n queries）语句。

现在您可以使用例 16-89 的 DCL 语句再将 cat 用户的 baby_cat 表的查询（SELECT）权限授予 fox 用户。

例 16-89

```
SQL> GRANT select
  2  ON     cat.baby_cat
  3  TO     fox;
```

例 16-89 结果

```
授权成功。
```

然后，您应该使用例 16-90 的 SQL*Plus 命令切换到 fox 用户。

例 16-90

```
SQL> connect fox/loveyou
```

例 16-90 结果

```
已连接。
```

别忘了使用例 16-91 的查询语句检查您所做的授权是否真的成功了。

例 16-91

```
SQL> SELECT rownum "Cat NO.", ename "Cat Name"
  2  FROM  (SELECT ename
  3         FROM cat.baby_cat
```

```
  4         ORDER BY ename)
  5  WHERE ROWNUM <= 6;
```

例 16-91 结果

```
  Cat NO. Cat Name
---------- --------
         1 ADAMS
         2 ALLEN
         3 BLAKE
         4 CLARK
         5 FORD
         6 JAMES
已选择 6 行。
```

为了测试您收回 WITH GRANT OPTION 选项授予的对象的权限会产生什么结果，您应该先使用例 16-92 的 SQL*Plus 命令切换到 cat 用户。

例 16-92

```
SQL> connect cat/miaomiao
```

例 16-92 结果

```
已连接。
```

然后，您可以使用例 16-93 的 DCL 语句从 pig 用户收回对 baby_cat 表的查询（SELECT）权限。

例 16-93

```
SQL> REVOKE select
  2  ON     baby_cat
  3  FROM   pig;
```

例 16-93 结果

```
撤销成功。
```

现在您可以开始对 pig、dog 和 fox 3 个用户分别进行测试，看看它们对 cat 用户 baby_cat 表的查询（SELECT）权限是否已被收回。

您可以使用例 16-94 的 SQL*Plus 命令先切换到 pig 用户。

例 16-94

```
SQL> connect pig/hengheng;
```

例 16-94 结果

```
已连接。
```

此时，您可以使用例 16-95 的查询语句来测试 pig 用户是否有权访问 cat 用户中的 baby_cat 表，即对 cat 用户中 baby_cat 表的查询（SELECT）权限是否已被收回。

例 16-95

```
SQL> SELECT *
  2  FROM cat.baby_cat;
```

例 16-95 结果

```
FROM cat.baby_cat
          *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

测试完了 pig 用户，下面您就可以测试 dog 用户了。您可使用例 16-96 的 SQL*Plus 命令先切换到 dog 用户。

例 16-96

```
SQL> connect dog/wangwang
```

例 16-96 结果

```
已连接。
```

现在，您可以使用例 16-97 的查询语句来测试 dog 用户是否有权访问 cat 用户中的 baby_cat 表，即对 cat 用户中 baby_cat 表的查询（SELECT）权限是否已被收回。

例 16-97

```
SQL> SELECT *
  2  FROM cat.baby_cat;
```

例 16-97 结果

```
FROM cat.baby_cat
          *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

最后，您就可以测试 fox（狐狸）用户了。您可使用例 16-98 的 SQL*Plus 命令先切换到 fox 用户。

例 16-98

```
SQL> connect fox/loveyou
```

例 16-98 结果

```
已连接。
```

现在，您可以使用例 16-99 的查询语句来测试 fox 用户是否有权访问 cat 用户中 baby_cat 表，即 cat 用户中 baby_cat 表的查询（SELECT）权限是否已被收回。

例 16-99

```
SQL> SELECT *
  2  FROM cat.baby_cat;
```

例 16-99 结果

```
FROM cat.baby_cat
           *
ERROR 位于第 2 行:
ORA-00942: 表或视图不存在
```

下面我们对以上实例中授权和回收权限的操作作如下的总结。

授予对象权限的操作：

（1）cat（猫）大哥（用户）以 WITH GRANT OPTION 的方式将他的 baby_cat 表的查询权限授予了 pig（猪）小弟（用户）。

（2）pig（猪）小弟（用户）又以 WITH GRANT OPTION 的方式将 cat（猫）大哥（用户）的 baby_cat 表的查询权限授予了他的 dog（狗）朋友（用户）。

（3）dog（狗）又将 cat（猫）大哥（用户）的 baby_cat 表的查询权限授予了他的 fox（狐狸）朋友（用户）。

回收对象权限的操作：

当 cat（猫）大哥（用户）从 pig（猪）小弟（用户）回收 baby_cat 表的查询权限后，pig（猪）小弟的狐朋狗友对 baby_cat 表的查询权限也自动消失（被收回）。

16.11　改变用户的口令

您可能还记得用户的初始口令（密码）是由 DBA 通过 CREATE USER 语句赋予的，这个口令可能不太安全。但是没关系，用户可以使用 ALTER USER 语句改变这一口令。

ALTER USER 语句的格式如下：

ALTER USER 用户名

IDENTIFIED BY 口令;

假设 cat（猫）大哥觉得他的口令 miaomiao 太不安全了，因为没谁不知道 miaomiao 是他发出的叫声。于是，他决定将他的口令改为 guagua（呱呱），真不愧为老“猫”深算。

为了修改 cat 用户的口令，您应该使用例 16-100 的 SQL*Plus 命令先切换到 cat 用户。

例 16-100

```
SQL> connect cat/miaomiao
```

例 16-100 结果

```
已连接。
```

现在您可以使用例 16-101 的 DDL 语句将 cat 用户的口令改为（鸭叫）guagua。

例 16-101

```
SQL> ALTER USER cat
  2  IDENTIFIED BY guagua;
```

例 16-101 结果

```
用户已更改。
```

您可以用如下的方法来测试您上面的操作是否成功。
首先，您可以使用例 16-102 的 SQL*Plus 命令切换到 dog 用户。

例 16-102

```
SQL> connect dog/wangwang
```

例 16-102 结果

```
已连接。
```

现在，您试着使用例 16-103 的 SQL*Plus 命令以原来的口令 miaomiao 连接到 cat 用户。

例 16-103

```
SQL> connect cat/miaomiao
```

例 16-103 结果

```
ERROR:
ORA-01017: invalid username/password; logon denied

警告：您不再连接到 Oracle。
```

例 16-103 显示的结果表明 cat 用户的口令已不是 miaomiao 了。
现在，您试着使用例 16-104 的 SQL*Plus 命令以新口令 guagua 连接到 cat 用户。

例 16-104

```
SQL> connect cat/guagua;
```

例 16-104 结果

```
已连接。
```

有时用户可能会忘了自己的口令，这时作为 DBA，您该怎么办呢？例如，fox 这个用户太精明了，所以总怕别人算计他，因此，三天两头改口令。终于，连他自己都记不得他的密码了，于是他来求您帮助。
您首先使用例 16-105 的 SQL*Plus 命令切换到 system（DBA）用户。

例 16-105

```
SQL> connect system/manager
```

例 16-105 结果

```
已连接。
```

现在您就可以使用例 16-106 的 DDL 语句将 fox 用户的口令也改为 fox（为了好记）。

例 16-106

```
SQL> ALTER USER fox
  2  IDENTIFIED BY fox;
```

例 16-106 结果

```
用户已更改。
```

现在 fox 用户就可以使用例 16-107 的 SQL*Plus 命令以 fox 密码登录系统了。

例 16-107

```
SQL> connect fox/fox;
```

例 16-107 结果

```
已连接。
```

当进入系统之后，fox 用户就可以再使用 ALTER USER 语句将他的口令重新修改成只有天知地知还有他知的新口令。希望他这次能记住这么“安全”的口令。

16.12　删 除 用 户

当某个用户不再需要使用系统时，如此人已离开公司，您可以从系统中删除该用户，如 fox 已经跳槽到公司的竞争对手那里去了。因此，您可以使用如下的 DDL 语句（例 16-109）从系统中删除 fox 用户。但在这之前您应该先使用例 16-108 的 SQL*Plus 命令先切换到 SYSTEM（DBA）用户。

例 16-108

```
SQL> connect system/manager;
```

例 16-108 结果

```
已连接。
```

例 16-109

```
SQL> DROP USER fox;
```

例 16-109 结果

```
用户已丢弃。
```

现在，您可以试着使用例 16-110 的 SQL*Plus 命令以原来的口令 fox 连接到 fox 用户。

例 16-110

```
SQL> connect fox/fox;
```

例 16-110 结果

```
ERROR:
ORA-01017: invalid username/password; logon denied

警告：您不再连接到 Oracle。
```

例 16-110 显示的结果表明，您已无法使用 fox/fox 连接到系统上，这间接地说明 fox

用户已被删除。

您也可以通过查询数据字典 dba_users 来得到更加准确的信息。但是，在这之前您还应该先使用例 16-111 的 SQL*Plus 命令先切换到 SYSTEM（DBA）用户。

例 16-111

```
SQL> connect system/manager
```

例 16-111 结果

```
已连接。
```

现在您就可以使用例 16-112 的查询语句来查看 fox 用户是否还在系统中。

例 16-112

```
SQL> SELECT username, created
  2  FROM dba_users
  3  WHERE ROUND(created, 'DAY') >= ROUND(SYSDATE, 'DAY') - 7;
```

例 16-112 结果

```
USERNAME                       CREATED
------------------------------ ----------
DOG                            29-4 月 -03
CAT                            29-4 月 -03
PIG                            30-4 月-03
```

例 16-112 显示的结果表明，您已经成功地删除了用户 fox。

注意：

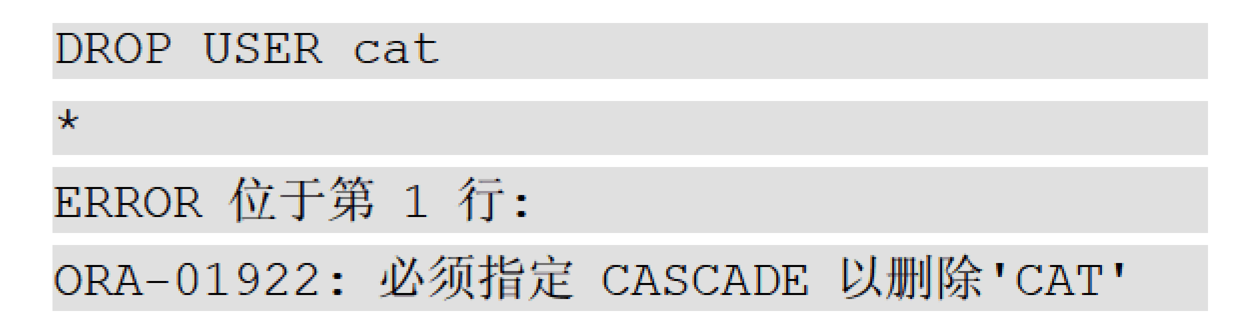
例 16-112 使用的 SQL 语句为显示一周之内所创建用户的用户名和创建日期。

您也可以使用例 16-113 的 DDL 语句试着从系统中删除 cat 用户。

例 16-113

```
SQL> DROP USER cat;
```

例 16-113 结果



```
DROP USER cat
*
ERROR 位于第 1 行:
ORA-01922: 必须指定 CASCADE 以删除'CAT'
```

看到例 16-113 显示的结果您马上意识到，您曾经在 cat 用户中创建过对象（baby_cat 表），而且它现在还存在于系统中，所以，您在 DROP USER 语句中要加上 CASCADE 选项。于是，您重新执行例 16-114 的 DDL 语句，再试着从系统中删除 cat 用户。

例 16-114

```
SQL> DROP USER cat CASCADE;
```

例 16-114 结果

```
用户已丢弃。
```

例 16-114 显示的结果表明，您已经成功删除了用户 cat。但为了慎重起见，您还是应该使用例 16-115 的查询语句来查看 cat 用户是否还在系统中。

例 16-115

```
SQL> SELECT username, created
  2  FROM dba_users
  3  WHERE ROUND(created, 'DAY') >= ROUND(SYSDATE, 'DAY') - 7;
```

例 16-115 结果

```
USERNAME                       CREATED
------------------------------ ----------
DOG                            29-4 月 -03
PIG                            30-4 月-03
```

看到例 16-115 显示的结果，您应当放心了。

当一个用户被删除之后，该用户中的所有对象也都从系统中消失了。就像该用户刚刚建立，没有任何对象一样。

注意：

删除用户是一个很危险的操作，因此，在这一操作之前最好做备份。虽然一个人离开了公司，但他/她存在系统中所有的东西都是属于公司的财产，不能因为他/她的离去而丢失。

16.13　CONNECT和RESOURCE角色

在所有的 Oracle 系统中都有两个预定义的角色，它们是 CONNECT 和 RESOURCE。尽管 Oracle 公司一再声称 CONNECT 和 RESOURCE 角色是为了与它早期的版本兼容而保留的，而且劝告用户尽可能不使用这两个角色以避免产生安全漏洞，但是，这两个角色到目前为止还是被广泛使用的。原因是利用这两个预定义的角色为用户授权非常简单。下面我们就介绍一下这两个角色。

首先，使用例 16-116 的 SQL*Plus 命令先切换到 SYS（DBA）用户。

例 16-116

```
SQL> CONNECT SYS/Oracle AS SYSDBA
```

例 16-116 结果

```
已连接。
```

然后，使用例 16-117 的查询语句查看 CONNECT 和 RESOURCE 这两个预定义的角色到底包含哪些系统权限。

例 16-117

```
SQL> SELECT *
  2  FROM role_sys_privs
  3  WHERE role IN ('CONNECT', 'RESOURCE');
```

例 16-117 结果

```
ROLE       PRIVILEGE                 ADM
---------- ------------------------- ---
CONNECT    ALTER SESSION             NO
CONNECT    CREATE CLUSTER            NO
CONNECT    CREATE DATABASE LINK      NO
CONNECT    CREATE SEQUENCE           NO
CONNECT    CREATE SESSION            NO
CONNECT    CREATE SYNONYM            NO
CONNECT    CREATE TABLE              NO
CONNECT    CREATE VIEW               NO

RESOURCE   CREATE CLUSTER            NO
RESOURCE   CREATE INDEXTYPE          NO
RESOURCE   CREATE OPERATOR           NO
RESOURCE   CREATE PROCEDURE          NO
RESOURCE   CREATE SEQUENCE           NO
RESOURCE   CREATE TABLE              NO
RESOURCE   CREATE TRIGGER            NO
RESOURCE   CREATE TYPE               NO
已选择 16 行。
```

一般不少公司的 DBA 把 CONNECT 角色授予所有的普通用户，而把 CONNECT 和 RESOURCE 两个角色授予所有的开发人员（程序员）。为了演示方便，您可以使用例 16-118 的 CREATE USER 语句重新创建用户 cat，该用户的密码仍为 miaomiao。

例 16-118

```
SQL> CREATE USER cat
  2  IDENTIFIED BY miaomiao;
```

例 16-118 结果

```
用户已创建。
```

现在，您可以使用例 16-119 的 DCL 语句将 CONNECT 角色授予 cat 这个用户。

例 16-119

```
SQL> GRANT CONNECT TO cat;
```

例 16-119 结果

```
授权成功。
```

然后，您可以试着利用 cat 这一用户登录系统。为了显示清楚，您可以使用例 16-120 的 SQL*Plus 命令。

例 16-120

```
SQL> connect cat/miaomiao;
```

例 16-120 结果

```
已连接。
```

此时，您可以通过使用数据字典 SESSION_PRIVS 来得到 cat 用户所拥有的全部系统权限的信息。您可以使用例 16-121 的查询语句来查看用户 cat 到底拥有多少系统权限。

例 16-121

```
SQL> SELECT *
   2  FROM SESSION_PRIVS;
```

例 16-121 结果

```
PRIVILEGE
--------------------
CREATE SESSION
ALTER SESSION
CREATE TABLE
CREATE CLUSTER
CREATE SYNONYM
CREATE VIEW
CREATE SEQUENCE
CREATE DATABASE LINK
已选择 8 行。
```

例 16-121 显示的结果表明，cat 用户所拥有的全部系统权限与例 16-117 显示结果中 CONNECT 角色所拥有的 8 个系统权限完全一样。

为了进一步演示 CONNECT 和 RESOURCE 两个角色的用法，您应该使用例 16-122 的 SQL*Plus 命令再次切换到 SYS（DBA）用户。

例 16-122

```
SQL> CONNECT SYS/Oracle AS SYSDBA
```

例 16-122 结果

```
已连接。
```

现在，您可以再次使用例 16-123 的 CREATE USER 语句重新创建用户 fox，该用户的密码为 devloper。

例 16-123

```
SQL> CREATE USER fox
   2  IDENTIFIED BY devloper;
```

例 16-123 结果

```
用户已创建。
```

然后，您可以使用例 16-124 的 DCL 语句将 RESOURCE 角色授予 fox 这个用户。

例 16-124

```
SQL> GRANT RESOURCE TO fox;
```

例 16-124 结果

```
授权成功。
```

然后，您可以试着用 fox 这一用户登录系统。为了显示清楚，您可以在 SQL*Plus 中使用例 16-125 的命令。

例 16-125

```
SQL> connect fox/devloper;
```

例 16-125 结果

```
ERROR:
ORA-01045: user FOX lacks CREATE SESSION privilege; logon denied
```

看到例 16-125 显示的结果之后，您马上重新查看例 16-117 的结果。您发现 RESOURCE 角色中并没有包含 CREATE SESSION 系统权限，所以 fox 用户无法登录系统。于是，您使用例 16-126 的 SQL*Plus 命令再次切换到 SYS（DBA）用户。

例 16-126

```
SQL> CONNECT SYS/Oracle AS SYSDBA
```

例 16-126 结果

```
已连接。
```

然后，您可以使用例 16-127 的 DCL 语句再将 CONNECT 角色也授予 fox 用户。

例 16-127

```
SQL> GRANT CONNECT TO fox;
```

例 16-127 结果

```
授权成功。
```

然后，您可以再试着利用 fox 这一用户登录系统。为了显示清楚，您可以在 SQL*Plus 中使用例 16-128 的命令。

例 16-128

```
SQL> CONNECT FOX/DEVLOPER;
```

例 16-128 结果

```
已连接。
```

此时，您就可以通过使用数据字典 SESSION_PRIVS 来得到 fox 用户所拥有的全部系统权限的信息。您可以使用例 16-129 的查询语句来查看用户 fox 到底拥有多少系统权限。

例 16-129

```
SQL> SELECT *
  2  FROM SESSION_PRIVS;
```

例 16-129 结果

```
PRIVILEGE
--------------------
CREATE SESSION
ALTER SESSION
UNLIMITED TABLESPACE
CREATE TABLE
CREATE CLUSTER
CREATE SYNONYM
CREATE VIEW
CREATE SEQUENCE
CREATE DATABASE LINK
CREATE PROCEDURE
CREATE TRIGGER
CREATE TYPE
CREATE OPERATOR
CREATE INDEXTYPE
已选择 14 行。
```

用了这么多笔墨来讲述 CONNECT 和 RESOURCE 这两个角色的目的并不是鼓励您使用这两个角色，而是为了让您了解这两个角色，因为实际的商业数据库管理中，还有不少的 DBA 在用它们。就像许多报纸和传媒大量地介绍“非典”一样，它们的目的绝不是希望读者和观众都染上“非典”，而是希望读者和观众更加了解这种疾病，从而更好地预防它。

如果您所工作的公司或机构的安全措施很差，公司的商业机密可以通过很多渠道轻易地得到（现实当中，许多公司的管理都是这样）。在这种情形下，数据库的安全措施已变得毫无意义。此时，您不妨通过 CONNECT 和 RESOURCE 这两个角色进行系统权限的管理，因为这样可以减少您不少工作负担。

使用 CONNECT 和 RESOURCE 这两个角色进行系统权限管理的另一个好处就是“快”。如果您自己定义角色，您首先要了解每个用户的操作特性，然后再根据他们的不同操作特性定义若干个角色，接着再把相应的系统权限授予相关的角色，最后再把这些角色授予相应的用户等。这样有时需要很长的时间。如果所用的时间太长，您的上司/老板可能会失去耐心。一般老板都希望雇来的人一上班就能干活，而且干得又快又好。这时 CONNECT 和 RESOURCE 这两个角色就很有用了，因为它们能使您干活快起来（至于好不好就只有天知道了，不过只要老板不知道就没事）。

16.14 应该掌握的内容

在学习完了这一章之后，请检查一下您是否已经掌握了以下的内容：

- 为什么要控制用户对数据库的访问？
- Oracle 数据库的安全管理包括什么？
- 如何创建用户及给用户赋口令？
- 如何利用数据字典 dba_users 获得用户的信息？
- Oracle 数据库管理系统中的权限分类。
- 什么是系统权限？
- 什么是对象权限？
- 什么是模式？
- 如何将系统权限授予用户？
- 如何获得用户所拥有的全部系统权限？
- 如何得到用户所具有的对象信息？
- 为什么要引入角色（Role）？
- 什么是角色（Role）？
- 如何创建角色（Role）？
- 如何将系统权限授予角色？
- 如何将角色分别授予多个用户？
- 如何得到一个角色所拥有的系统权限？
- 如何获得用户被授予的角色？
- 有哪 8 种对象的权限？
- 对象的权限和对象之间的关系。
- 如何将对象的权限授予其他用户？
- 如何获得对象和对象权限之间的关系？
- GRANT 语句中 WITH GRANT OPTION 选项的作用。
- 使用 WITH GRANT OPTION 选项可能带来的安全问题。
- 如何回收权限？
- 一般用户如何改变用户自己的口令？
- DBA 如何改变其他用户的口令？
- 如何从系统中删除用户？

第17章

图形工具简介和集合操作

在前面的所有章节中，我们都是通过命令行工具 SQL*Plus 演示 Oracle SQL 的操作的。这是因为 SQL*Plus 是所有 Oracle 版本必带的，而且也是最稳定的一个工具。有时在 Oracle 数据库出了问题时，它就成了唯一能使用的工具，即成了可以救活数据库的最后的“一根稻草”。因此，这个工具特别受到数据库管理员的青睐。

但是，一些从未接触过命令行的初学者往往觉得图形工具更亲切，也更容易掌握。另外，在进行较大型的软件开发时往往需要使用一些比较复杂的图形开发工具来提高软件开发的效率。人类之所以能进化成今天的万物之灵就是因为我们的祖先学会了发明和使用工具，虽然与其他动物相比，人类几乎没有什么长处，但是在工具的帮助下，最终人类成为了这个世界的主宰。图 17.1 为人类进化的示意图。

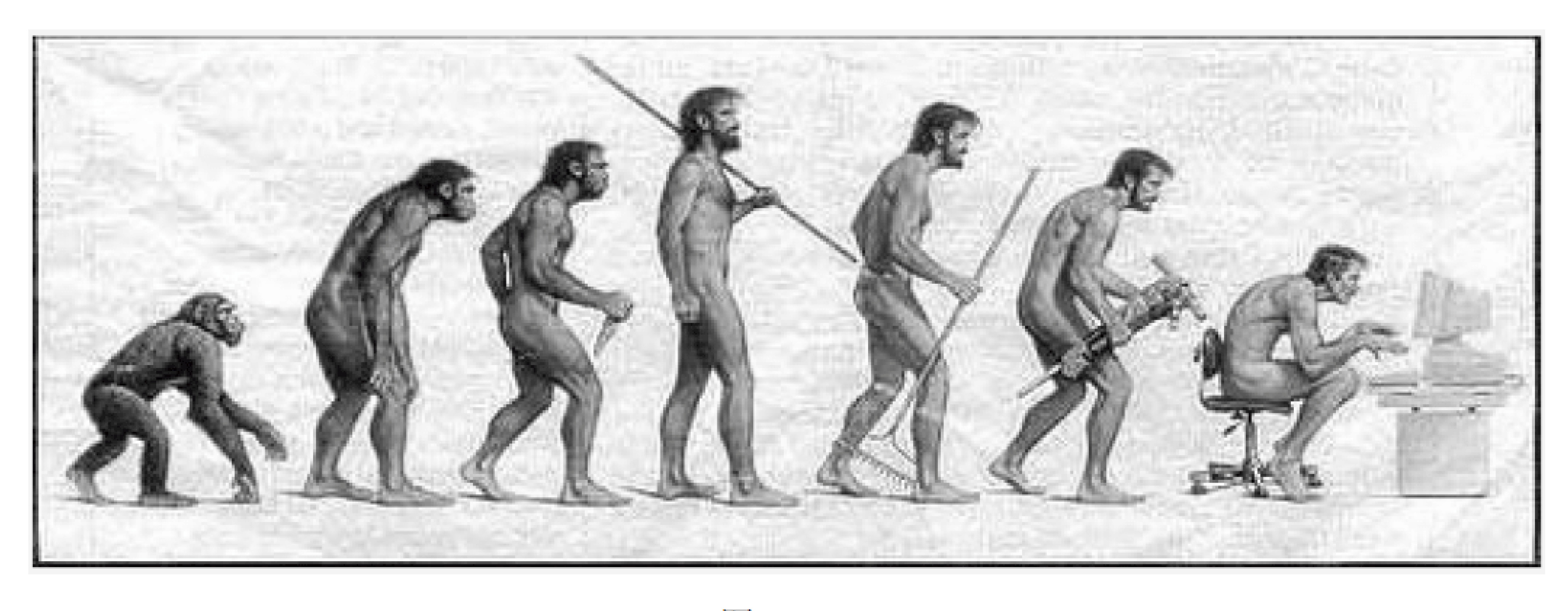


图 17.1

如果您将来要从事开发（编程）工作，您也需要学会使用复杂而功能强大的图形开发工具。借助于工具，您就可以像我们的祖先一样不断进化，最后成为一代宗师。下面我们将介绍两种比较常用的 Oracle 图形开发工具。

17.1 PL/SQL Developer 简介

PL/SQL Developer 是由荷兰的 Allround Automations 公司开发的一个图形工具，这是一个为 Oracle 开发人员提供的工具。Allround Automations 公司除了 PL/SQL Developer 之外，还开发了一些其他的 Oracle 工具。如果读者对这方面有兴趣，可以登录该公司的网站，其

网址为 http://www.allroundautomations.com。该工具不是免费的，价格是单用户 180 美元、5 个用户 540 美元、10 个用户 900 美元等（在 Oracle 的开发工具中是比较便宜的）。但是该公司提供免费的试用版，试用期一般为一个月。

PL/SQL Developer 的安装非常简单，只要用鼠标左键双击它的安装程序，然后在提示时输入安装码，然后根据提示一直单击“下一步”按钮即可。如果运气好的话，应该能一次安装成功。为了节省篇幅，这里就不给出具体的操作步骤了。PL/SQL Developer 是一个比较小的 Oracle 开发工具，它的 7.0.2 版安装程序还不到 29MB。

启动 PL/SQL Developer 的方法为：选择“开始”→“程序”→PL/SQL Developer→PL/SQL Developer 命令，如图 17.2 所示。

也可以通过使用鼠标左键双击桌面上的 PL/SQL Developer 图标来启动 PL/SQL Developer，如图 17.3 所示。

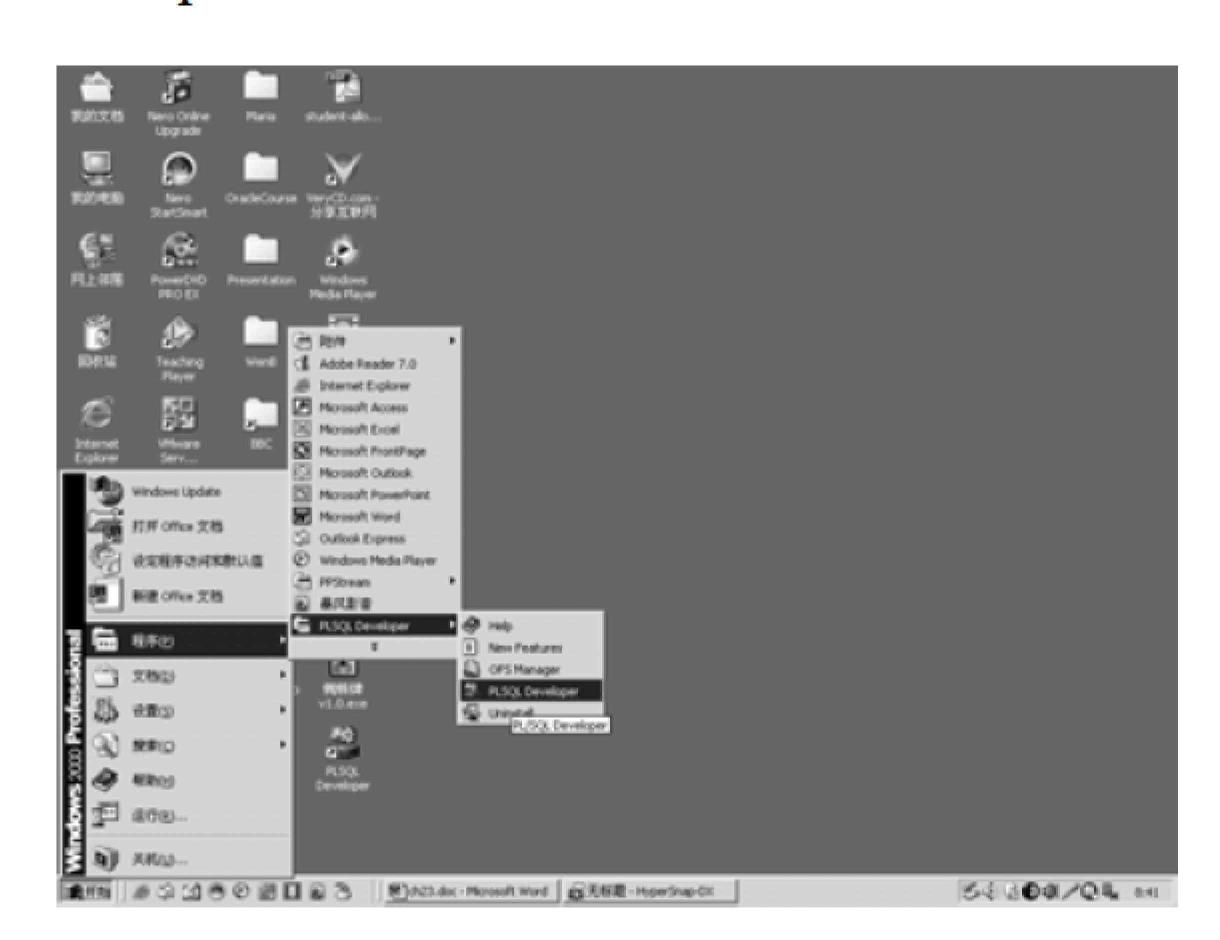


图 17.2



图 17.3

启动后，就会出现 PL/SQL Developer 的登录画面。此时输入用户名和密码等即可登录数据库系统，这里使用了读者十分熟悉的 SCOTT 用户，如图 17.4 所示。

单击 OK 按钮，系统就会使用 SCOTT 用户登录。当登录成功之后，在窗口左侧的下拉列表框中显示的是 All objects 选项，您需要选择 My objects 选项，否则将来显示的对象会太多，如图 17.5 所示。

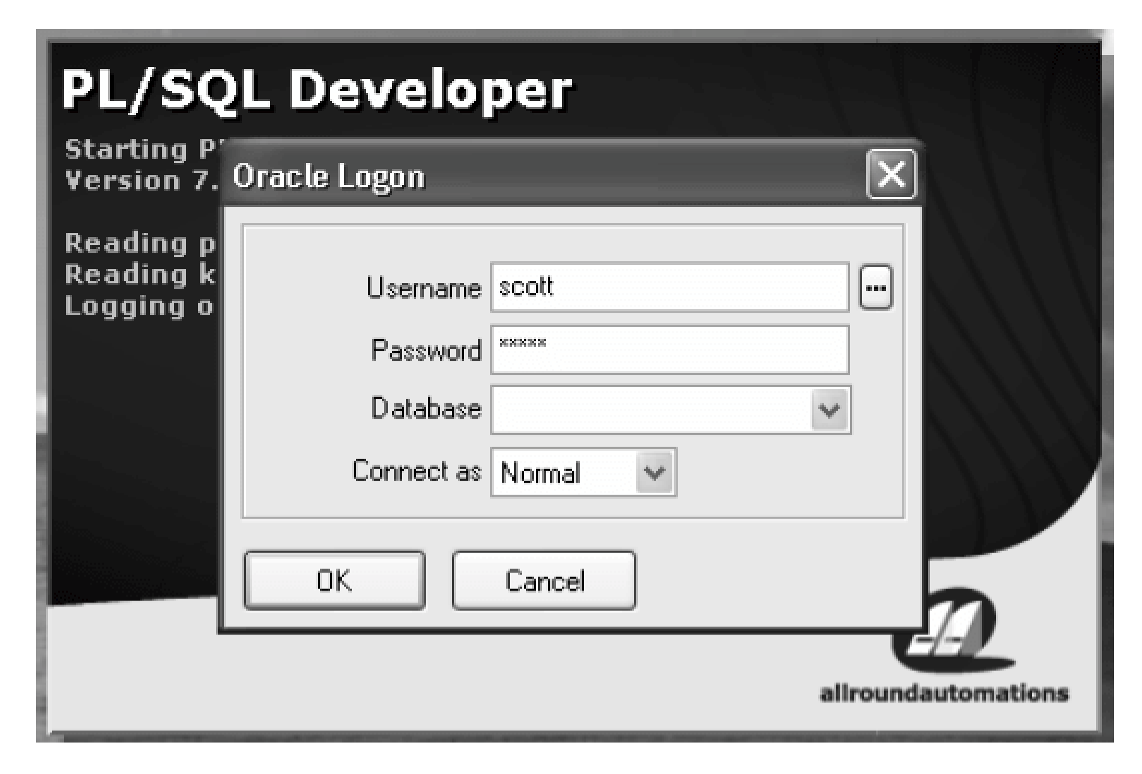


图 17.4

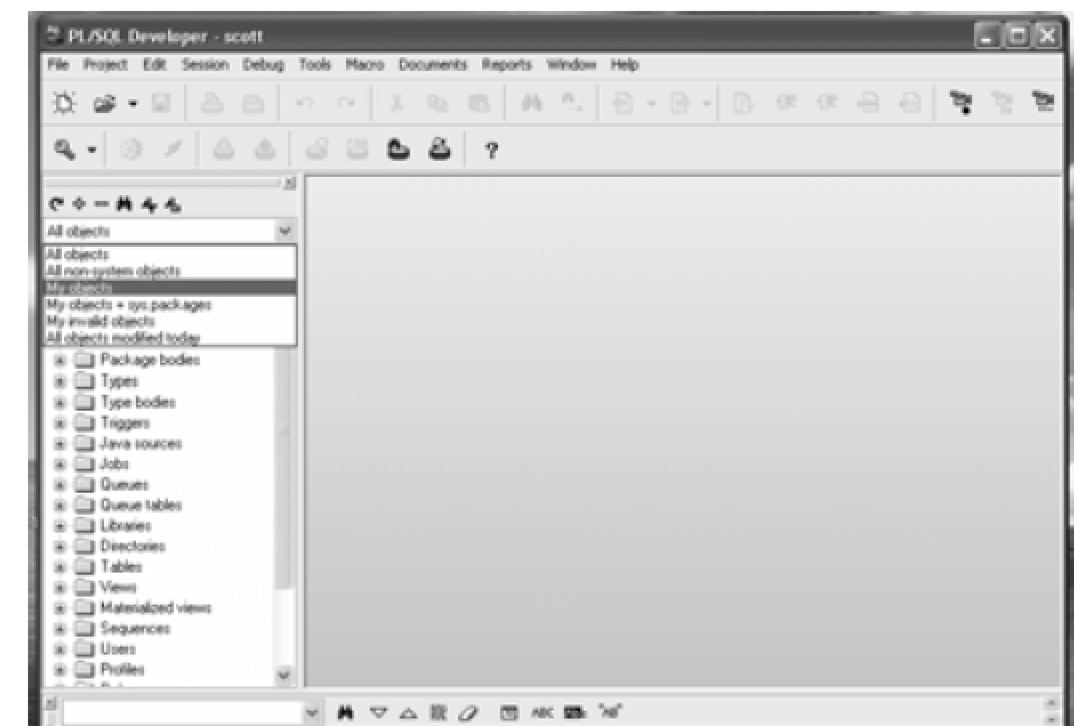


图 17.5

为了导出表的结构，应先选择表，然后使用鼠标右键单击 EMP 表，在弹出的快捷菜单中选择 View 命令，如图 17.6 所示。

然后就可以看到如图 17.7 所示的窗口，其中显示了 emp 表的存储结构的信息。接下来您可以选择 Columns 选项卡来获取该表的每一列的详细信息。还可以选择其他选项卡以获取需要的信息。为了减少显示输出，我们已经对显示输出进行了剪裁和调整。

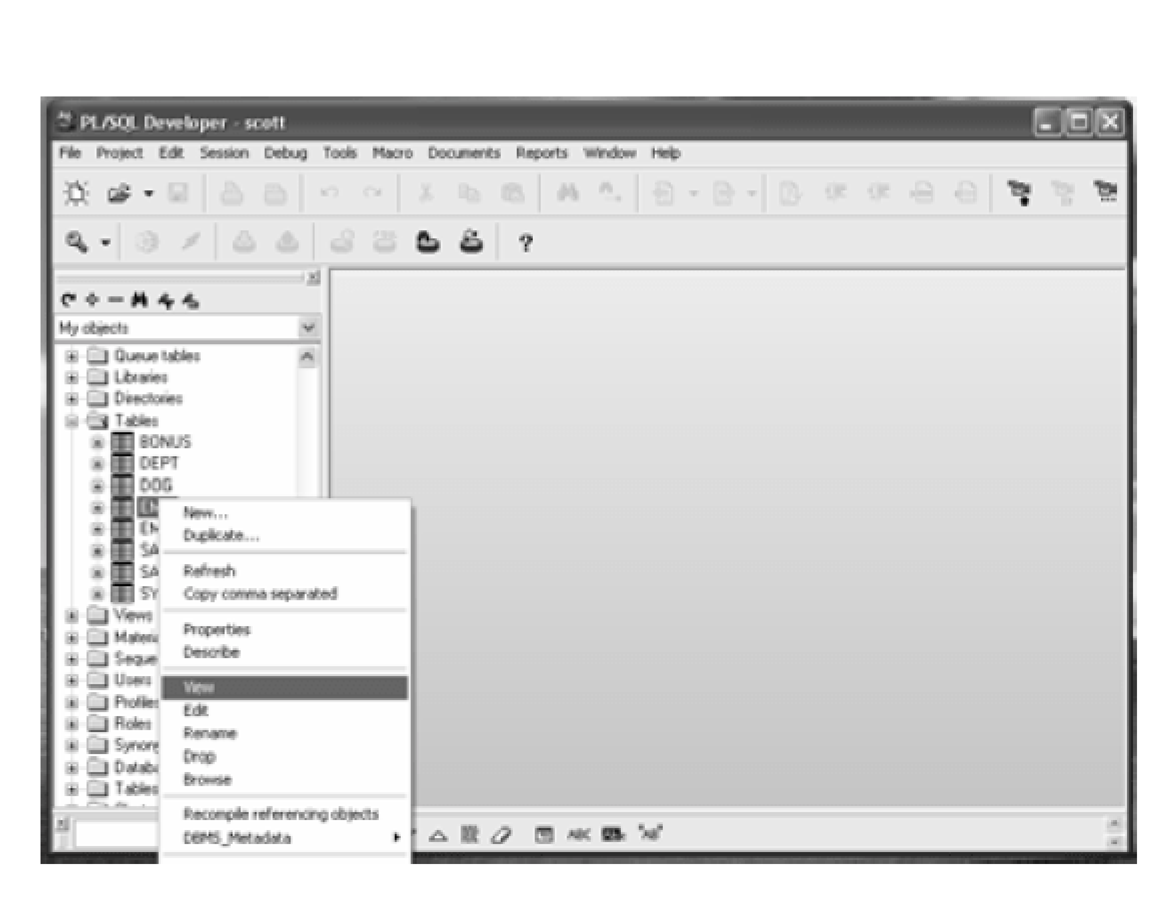


图　17.6

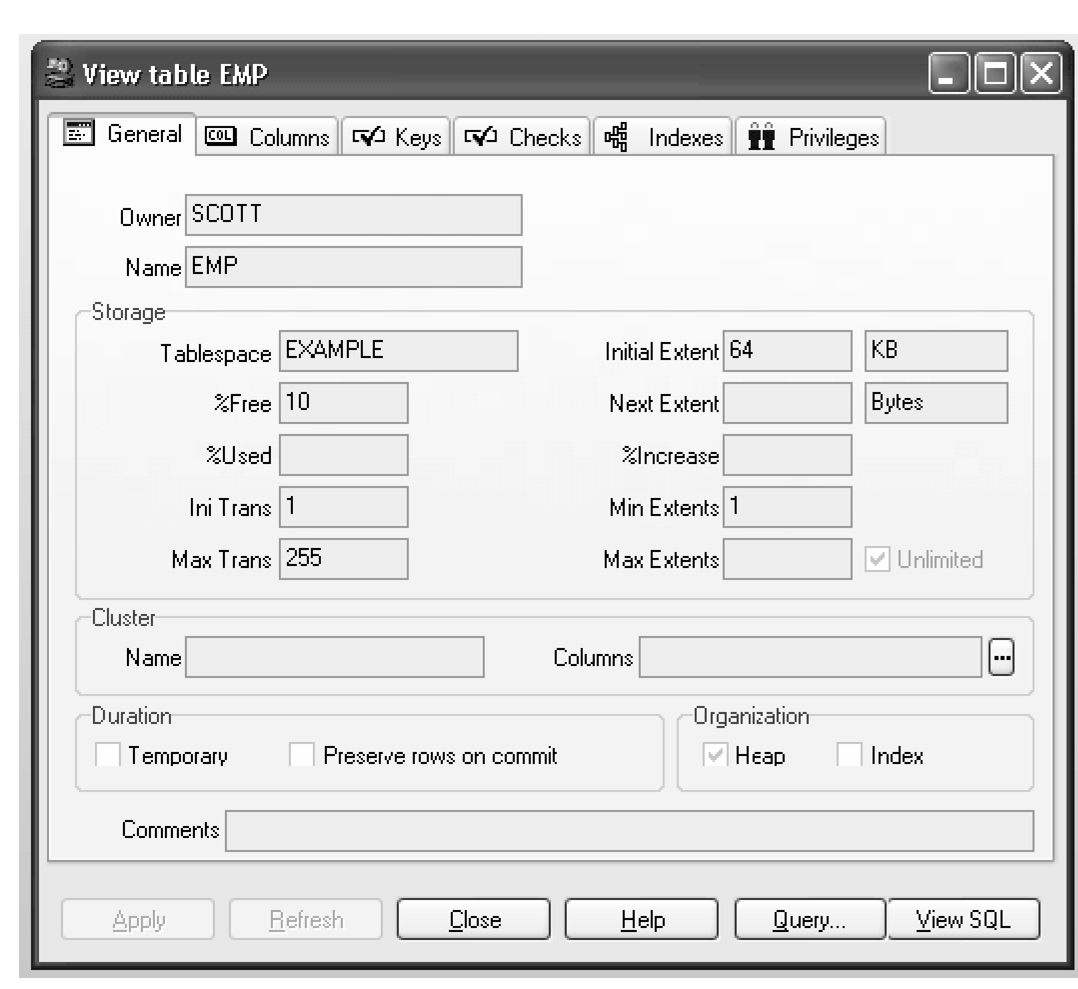


图　17.7

如果选择 Keys 选项卡，PL/SQL Developer 就会显示 emp 表的所有键，其中当然包括主键和外键，如图 17.8 所示。

也可以单击 View SQL 按钮，PL/SQL Developer 会显示创建 emp 表所用的 DDL 语句。如果您想将这些 SQL 语句导出，可以单击“保存”按钮，如图 17.9 所示。

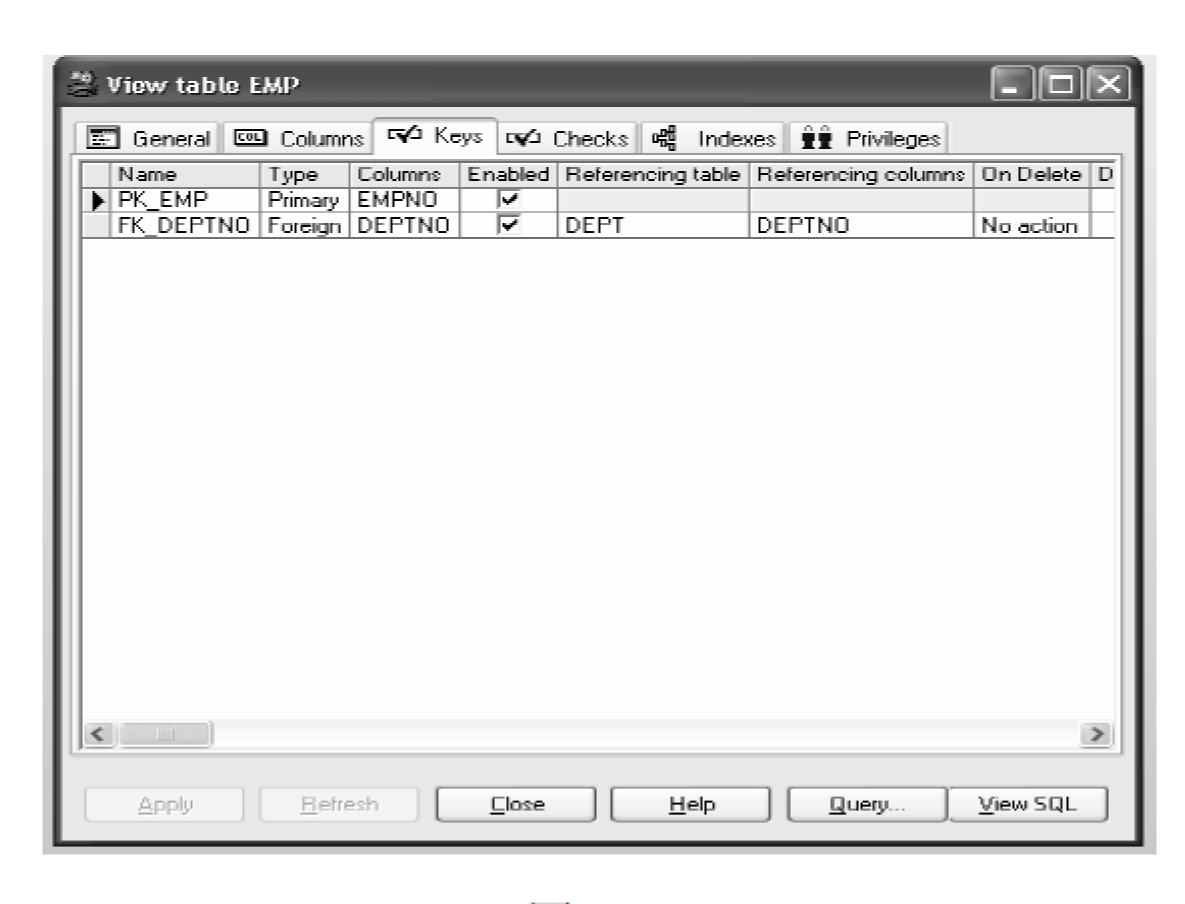


图　17.8

```
-- Create table
create table EMP
(
  EMPNO    NUMBER(4) not null,
  ENAME    VARCHAR2(10),
  JOB      VARCHAR2(9),
  MGR      NUMBER(4),
  HIREDATE DATE,
  SAL      NUMBER(7,2),
  COMM     NUMBER(7,2),
  DEPTNO   NUMBER(2)
)
tablespace USERS
  pctfree 10
  initrans 1
  maxtrans 255
  storage
  (
    initial 64K
    minextents 1
    maxextents unlimited
  );
-- Create/Recreate primary, unique and foreign key constraints
alter table EMP
  add constraint PK_EMP primary key (EMPNO)
```

图　17.9

利用以上所介绍的方法可以很轻松地使用 PL/SQL Developer 导出表的结构和表与表之间的关系（Keys），利用这些信息很容易地还原数据库的逻辑设计（实体-关系图）和物理设计（存储结构的设计）。是不是比使用 Oracle 的数据字典方便多了？

下面我们介绍怎样使用 PL/SQL Developer 导出 Oracle 的储存程序。为此，要以 HR 用户重新登录数据库系统，因为在该用户中有一些存储过程和函数，如图 17.10 所示。

☞ **指点迷津：**

如果 HR 用户无法登录，您需要将这个用户解锁。出于安全的考虑，从 Oracle 10g 开始，除 SYS 和 SYSTEM 之外的所有默认用户在 Oracle 数据库安装之后自动锁住。此时，要以 SYS 或 SYSTEM 用户登录 Oracle 数据库管理系统，然后使用 alter user hr identified by hr account unlock 命令打开 HR 的锁；这里第 2 个 hr 是口令（这是一个不安全的口令，您可以使用其他更安全的口令，这里使用只是为了操作方便）。

登录成功之后，选择 My objects 选项，然后展开 Procedures 节点，再用鼠标右键选择一个过程（这里选择 RAISE_SALARY，您也可以选择其他的过程），在弹出的快捷菜单中选择 View 命令，如图 17.11 所示。



图　17.10

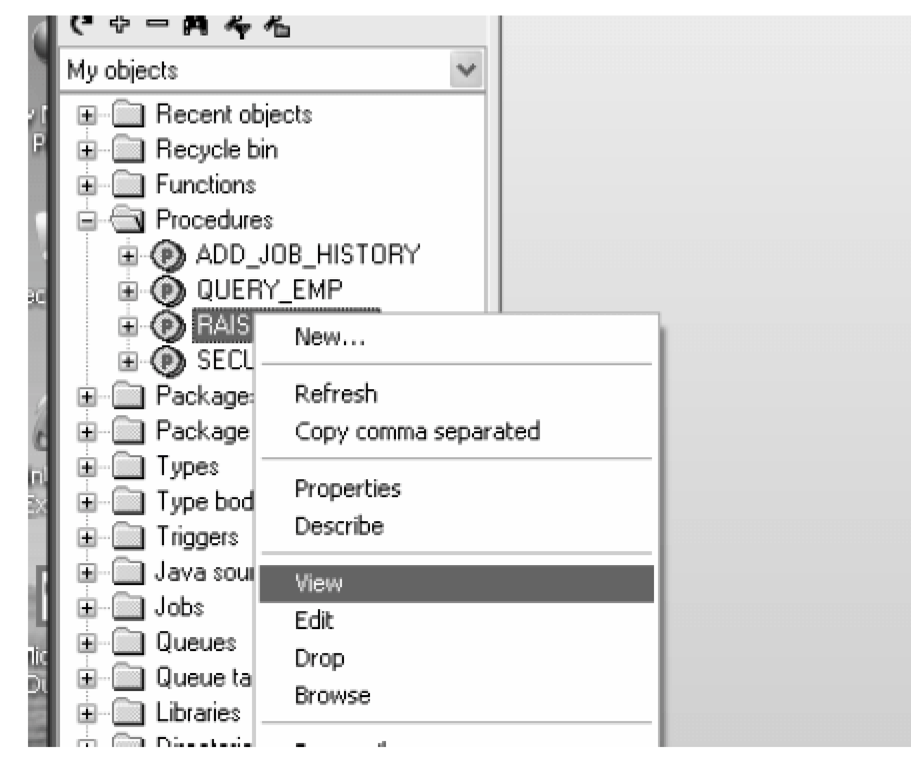


图　17.11

然后，PL/SQL Developer 会显示过程 RAISE_SALARY 的全部源代码，如图 17.12 所示。选择 File→Save As 命令就可以将源代码存入一个文件中，如图 17.13 所示。



图　17.12

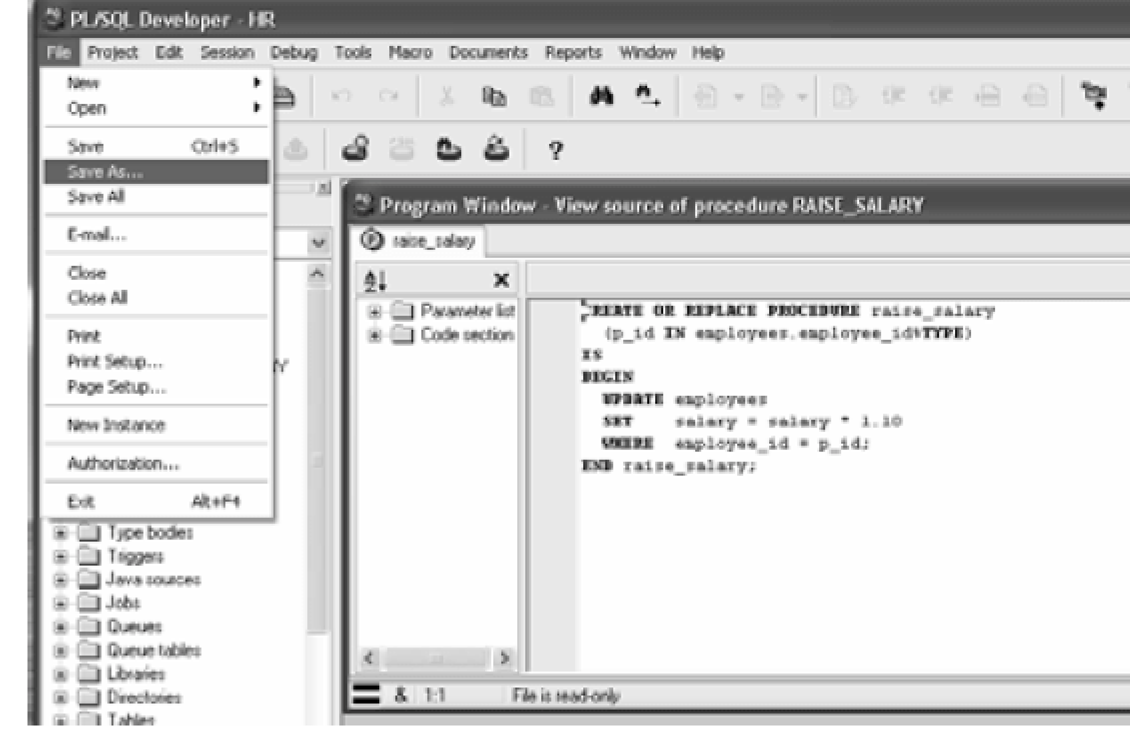


图　17.13

也可以使用与图 17.11 相似的操作导出存储函数 GET_SAL（也可以选择其他的存储函数）的源代码，如图 17.14 所示。因为所使用的操作十分相似，所以不再赘述。

接下来我们介绍如何利用 PL/SQL Developer 导出 SQL 语句的执行计划。

（1）选择 Session→Log on 命令，如图 17.15 所示。

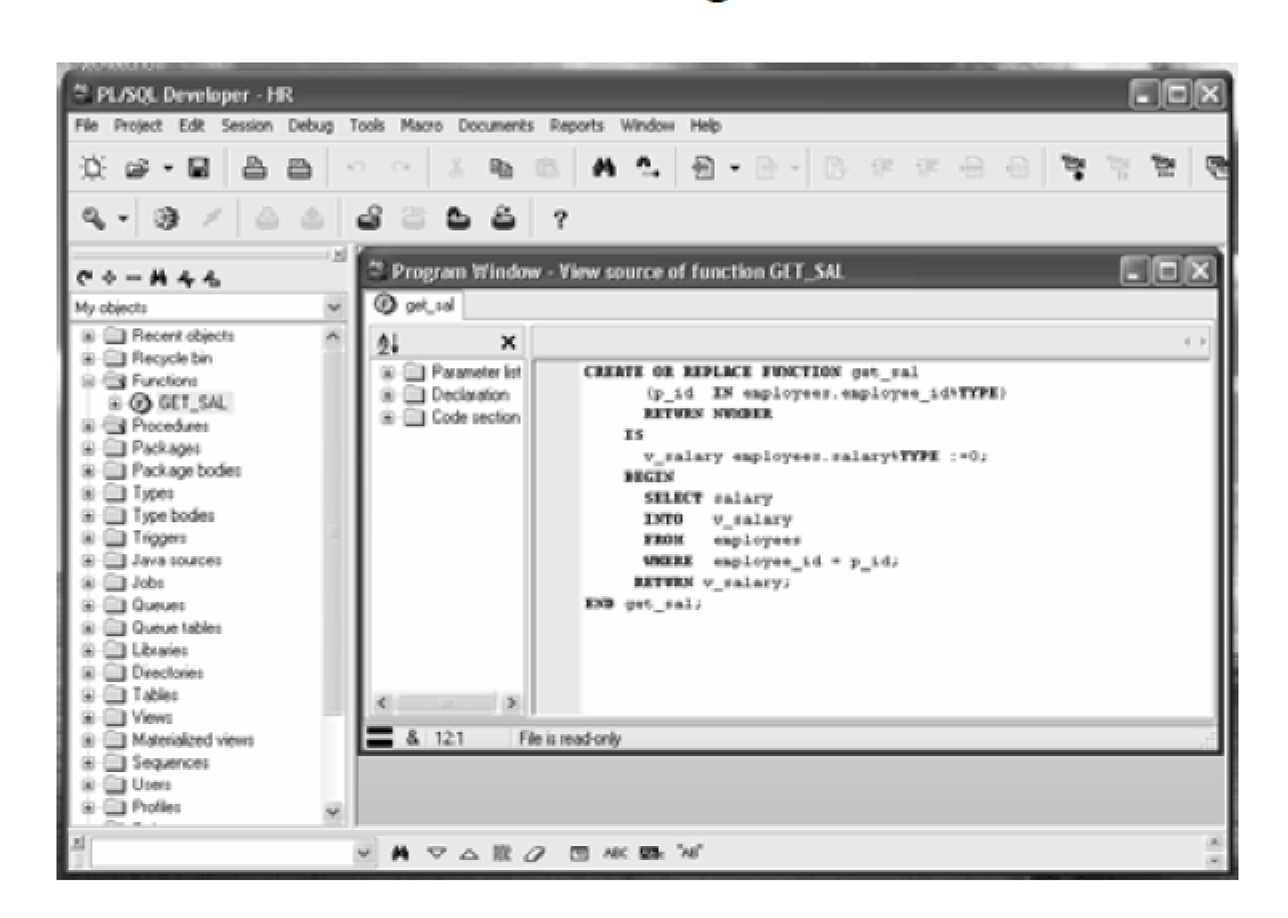


图　17.14

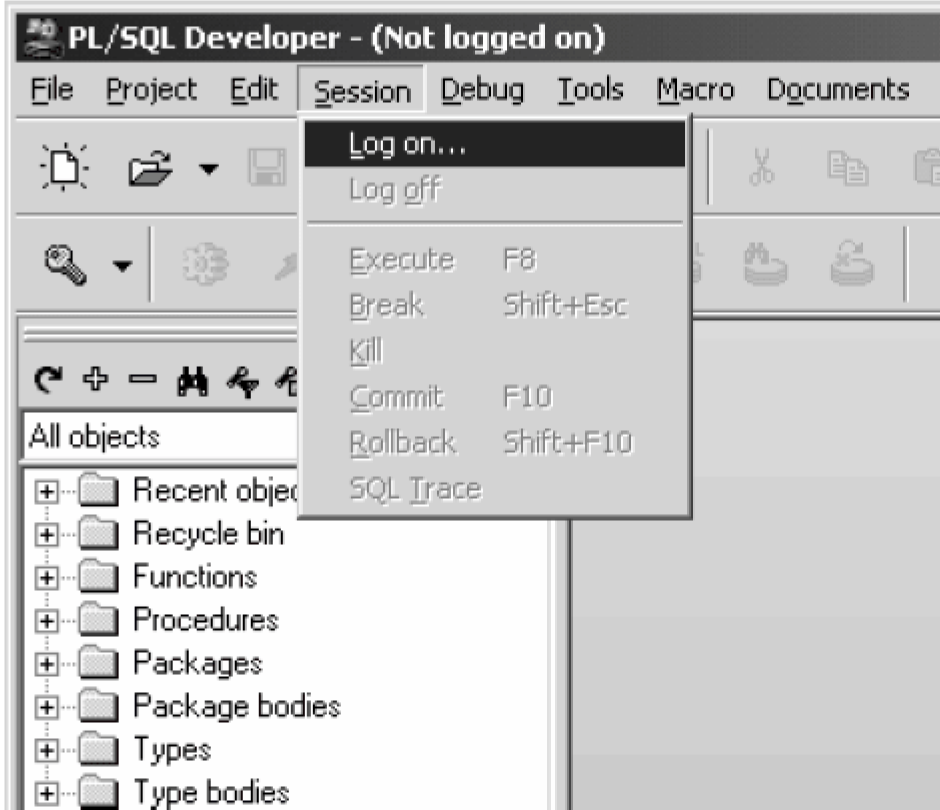


图　17.15

（2）在登录画面中输入用户名（scott）和密码（tiger），最后单击 OK 按钮，如图 17.16 所示。

（3）选择 File→New→SQL Window 命令，如图 17.17 所示。

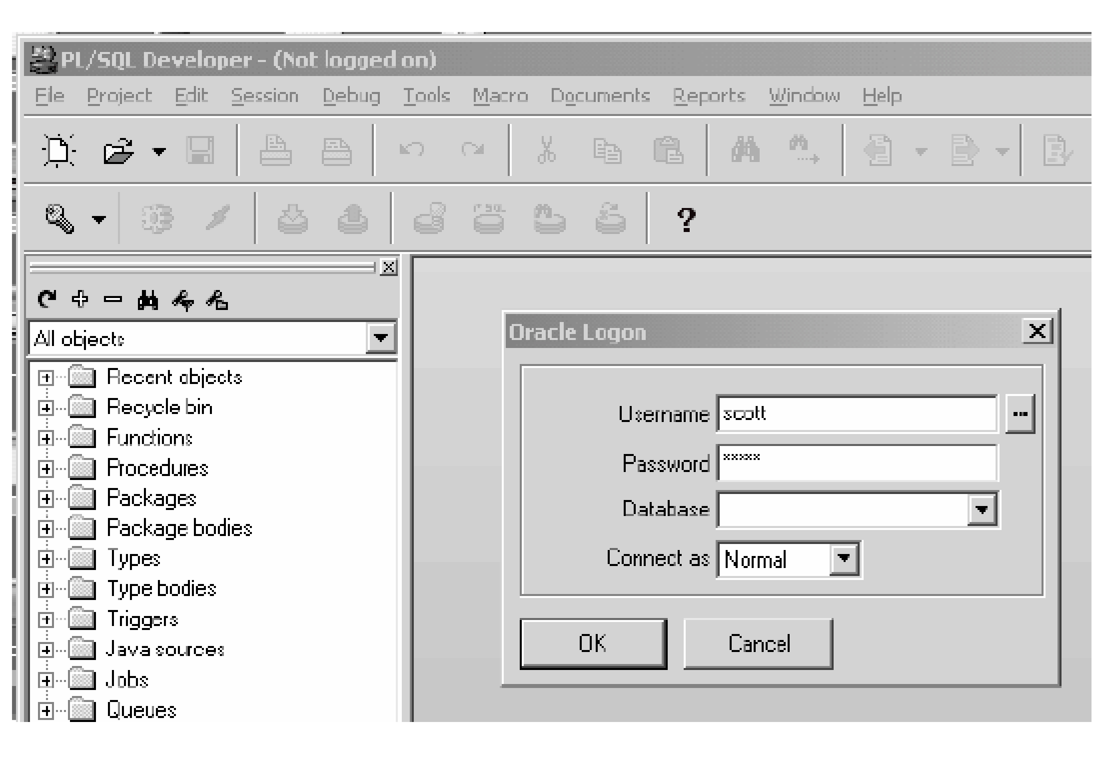


图　17.16

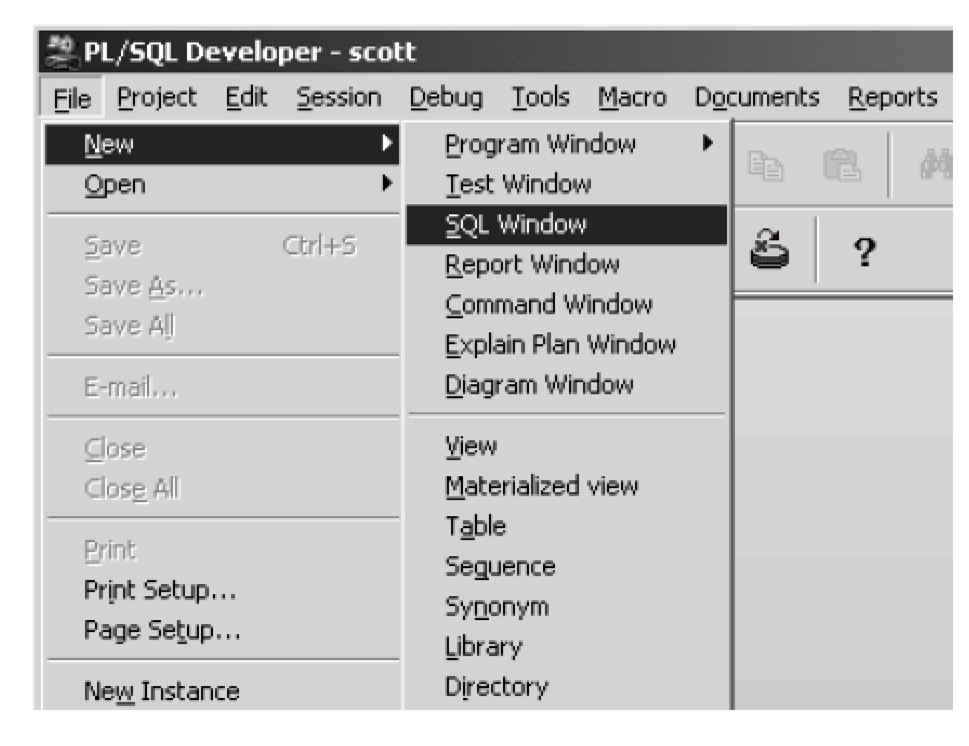
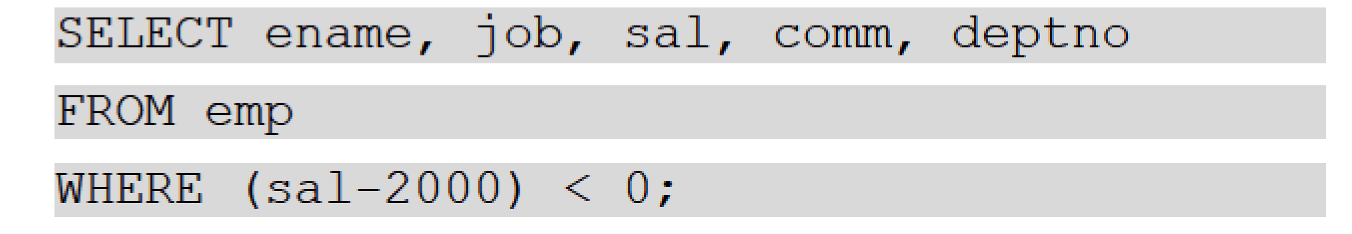


图　17.17

（4）当新的 SQL Window（窗口）打开之后，在里面输入如下的 SQL 查询语句：



```
SELECT ename, job, sal, comm, deptno
FROM emp
WHERE (sal-2000) < 0;
```

我们的目的是为了检验这个查询是否使用曾经创建的基于表达式 sal−2000 的索引，其画面如图 17.18 所示。

此时，可以通过单击“执行”图标（小齿轮形状的图标）或按 F8 键来执行 SQL Window（窗口）中的 SQL 语句。也可以通过单击“解释计划”图标或按 F5 键来获取 SQL Window（窗口）中 SQL 语句的执行计划，如图 17.19 所示。

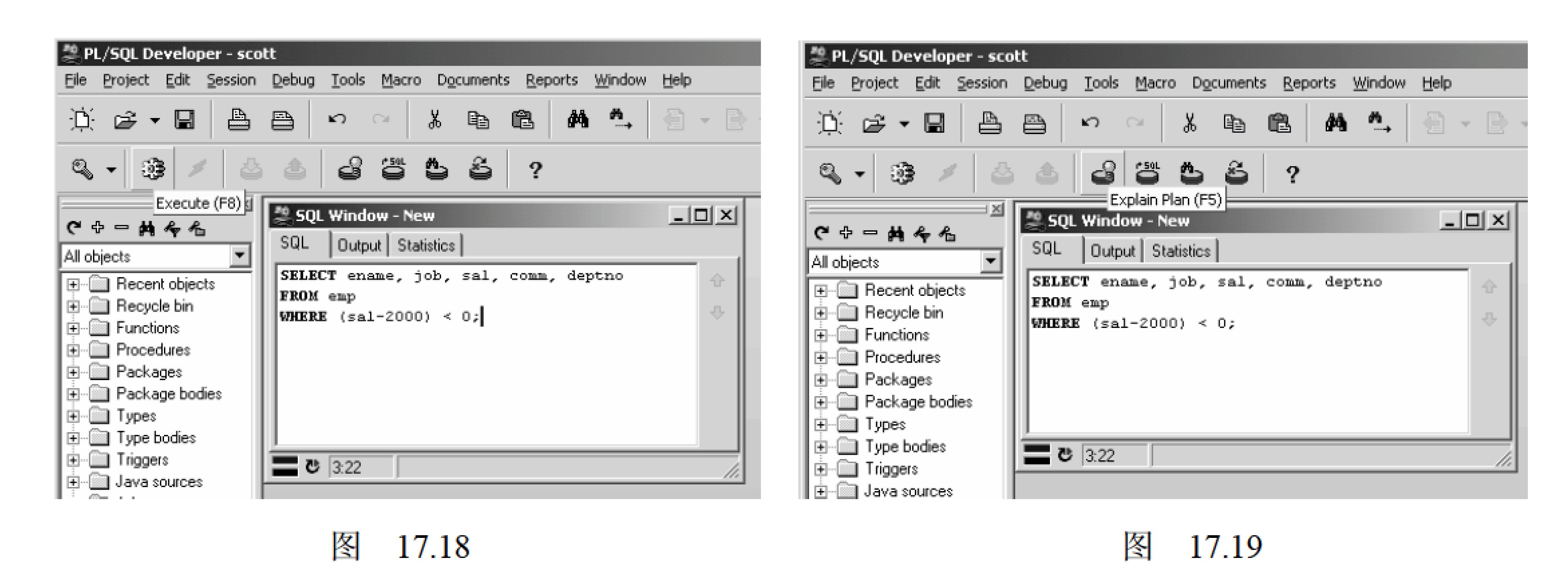


图 17.18　　　　图 17.19

（5）最后就获得了 SQL Window（窗口）中 SQL 语句的执行计划，如图 17.20 所示。该执行计划表明所解释的 SQL 语句确实使用了曾经创建的基于函数的索引 EMP_SALGT_IDX。

除了以上所介绍的功能之外，PL/SQL Developer 还包括了许多程序开发和调试的功能。利用这样的工具进行软件的开发可以在很大程度上提高软件的生产率，也可以适当地提高软件的质量。PL/SQL Developer 自带一个十分详细的用户手册（有 200 多页），可以通过如下方式获得：选择 Help→User's Guide 命令，如图 17.21 所示。

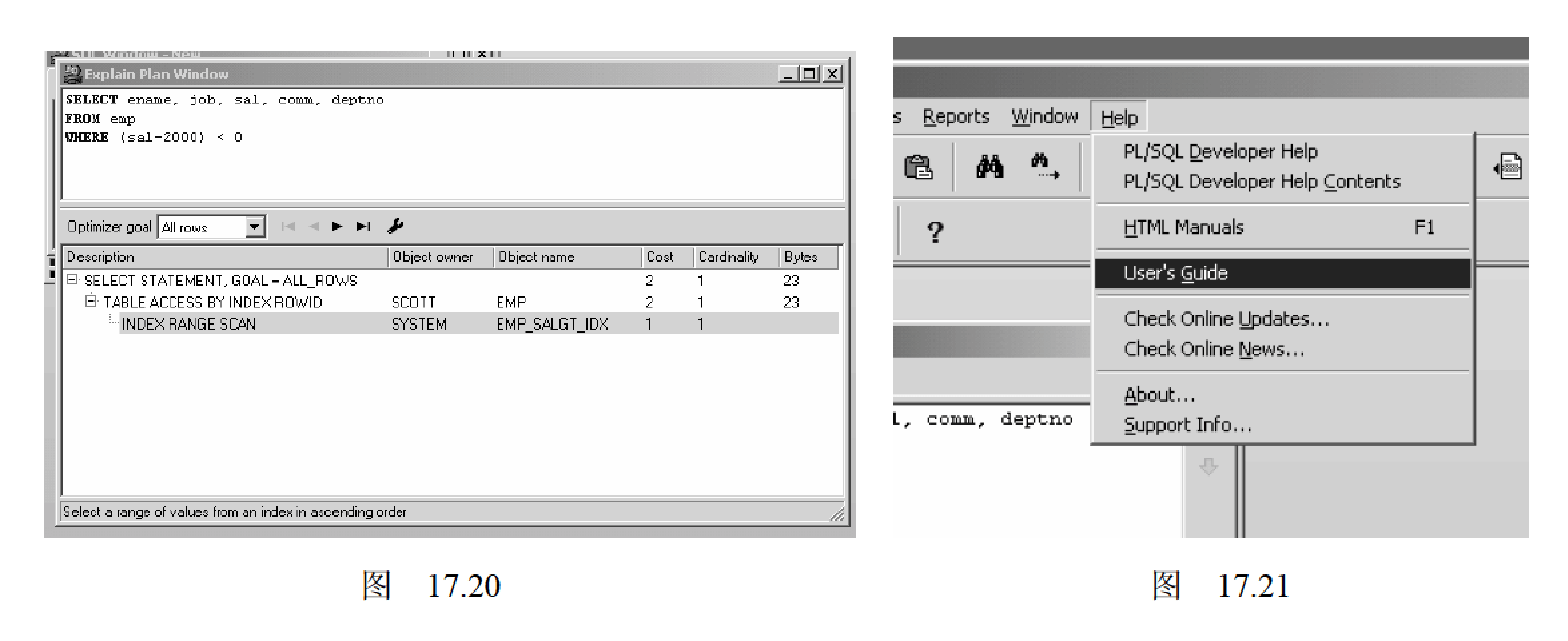


图 17.20　　　　图 17.21

17.2　Oracle SQL Developer 简介

Oracle SQL Developer 是 Oracle 公司不久之前推出的一个图形化的开发工具，它支持 Oracle 9.2.0.1 或以上的所有 Oracle 的版本。这个工具是免费的，可以在以下的网址免费下载：http://www.oracle.com/technology/products/database/sql_developer。另外，这个工具不需要安装，只要将下载的 Oracle SQL Developer 套件解压缩之后，就可以直接运行并使用。Oracle SQL Developer 是使用 Java 开发的，它支持 Windows、Linux 和 Mac 操作系统的 X 平台。Oracle SQL Developer 既可以直接与数据库服务器连接，也可以从远程的桌面系统连接到数据库系统。

Oracle SQL Developer 还可以直接连接到第三方的数据库模式（用户），如 MySQL、Microsoft SQL Server 和 Microsoft Access 等。本章使用的是 Oracle SQL Developer 的 1.2.1 版。

☞ **指点迷津：**

在下载 Oracle SQL Developer 套件时，最好下载带有 JDK1.5 的套件。因为 Oracle SQL Developer 运行时需要 JDK1.5，否则您可能需要单独安装 JDK1.5。

在将 Oracle SQL Developer 套件解压缩之后，该套件所有的文件都存放在 sqldeveloper 目录中，在该目录中的 sqldeveloper.exe 就是 Oracle SQL Developer 执行程序。为了以后使用方便，以如图 17.22 所示的方法将它发送到桌面上：用鼠标右键单击该程序，在弹出的快捷菜单中选择“发送到”→“桌面快捷方式”命令。

这样，就可以在桌面上看到 sqldeveloper.exe 的图标了，如图 17.23 所示。如果觉得该图标的名字不合适可以修改。

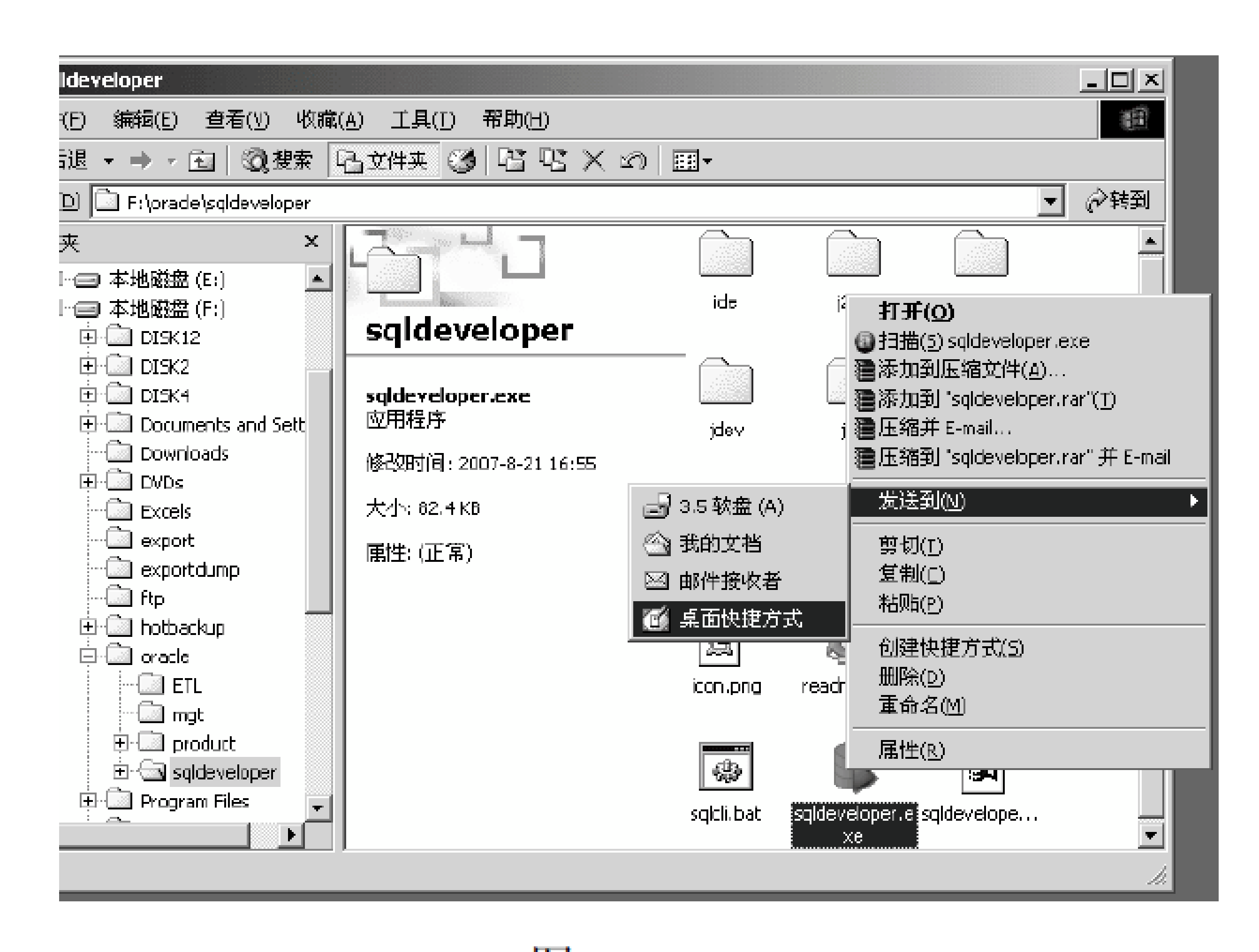


图　17.22



图　17.23

当使用鼠标左键双击 sqldeveloper.exe 的图标之后，应该出现如图 17.24 所示的画面。

当运行一会儿之后，就会出现 Oracle SQL Developer 的连接画面。此时，用鼠标右键单击 Connections，在弹出的快捷菜单中选择 New Connection 命令，如图 17.25 所示。

然后就会出现 Oracle SQL Developer 建立连接的画面。在此，我们为 SCOTT 用户建立一个连接：在 Connection Name 文本框中输入“SCOTT”；在 Username 文本框中输入“SCOTT”；在 Password 文本框中输入该用户的密码“TIGER”；为了以后操作方便，可以选中 Save Password 复选框，但这样却留下了安全隐患；SID 为实例名，该系统为 jinlian（您的系统可能不同）；其他可以使用默认值（如果监听进程使用的端口不是 1521，要改成所使用的端口号），如图 17.26 所示。

单击 Connect 按钮后，如果连接成功就会出现如图 17.27 所示的画面。这里要注意的是，Oracle SQL Developer 在建立连接时，要求监听进程必须已经启动，而 PL/SQL Developer 则并不要求这一点（在本机上使用时）。如果监听进程没有启动，Oracle SQL Developer 是无法建立连接的。

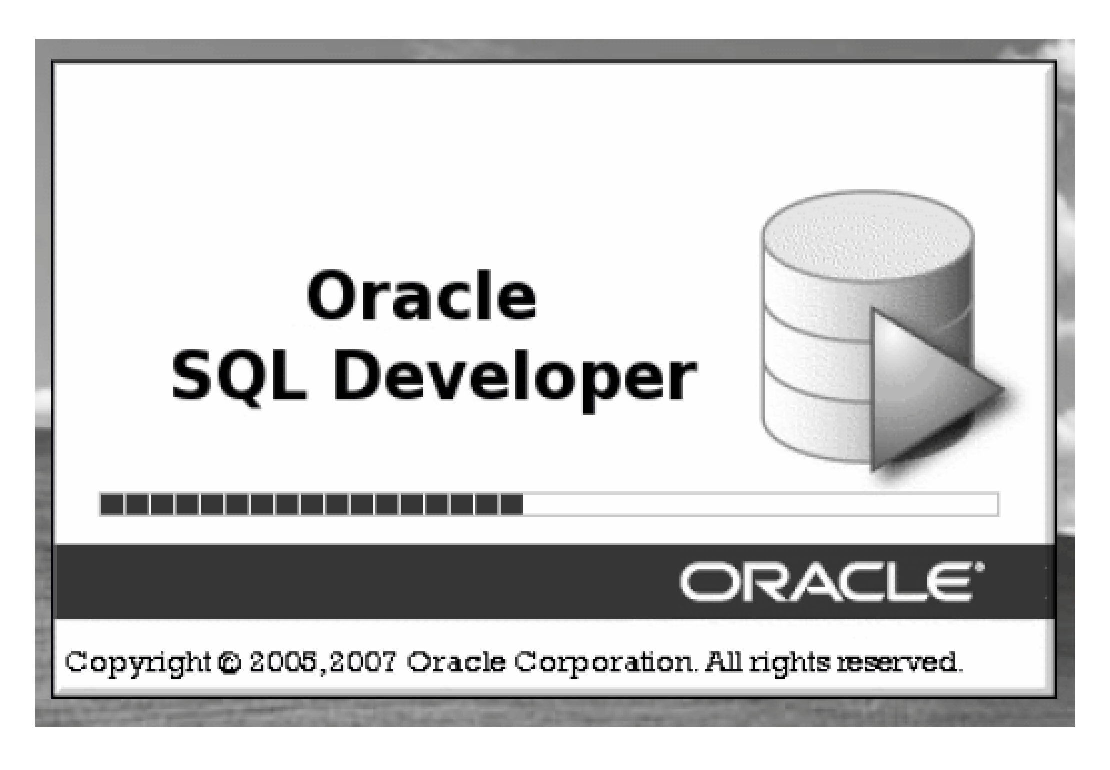


图　17.24

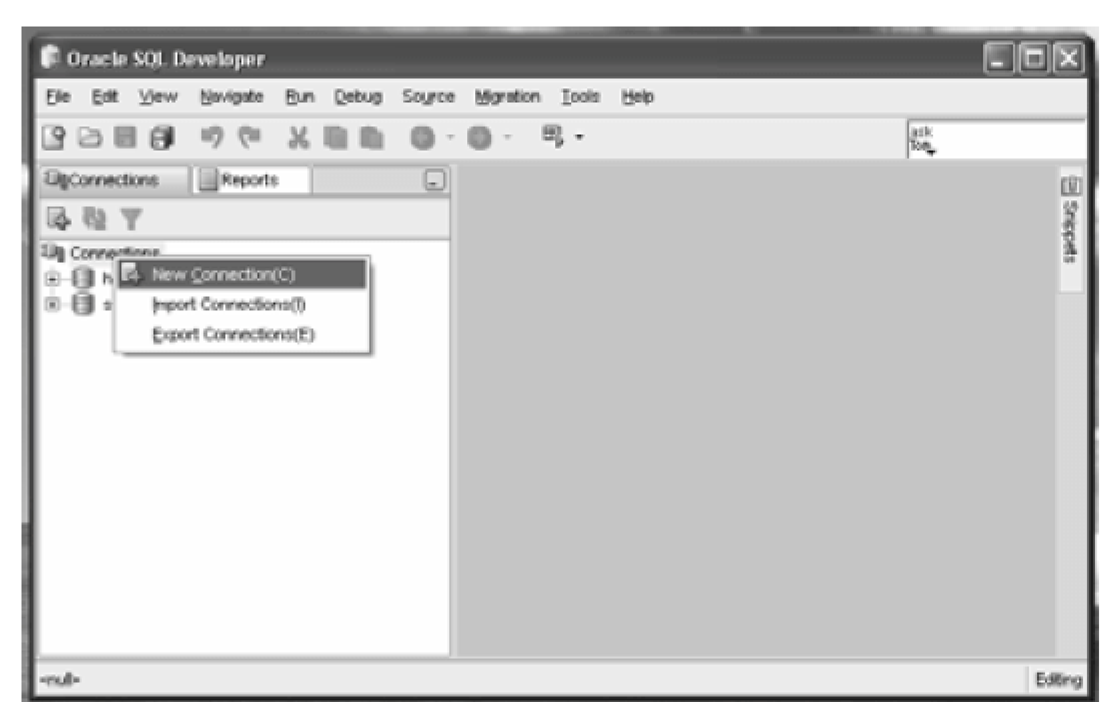


图　17.25

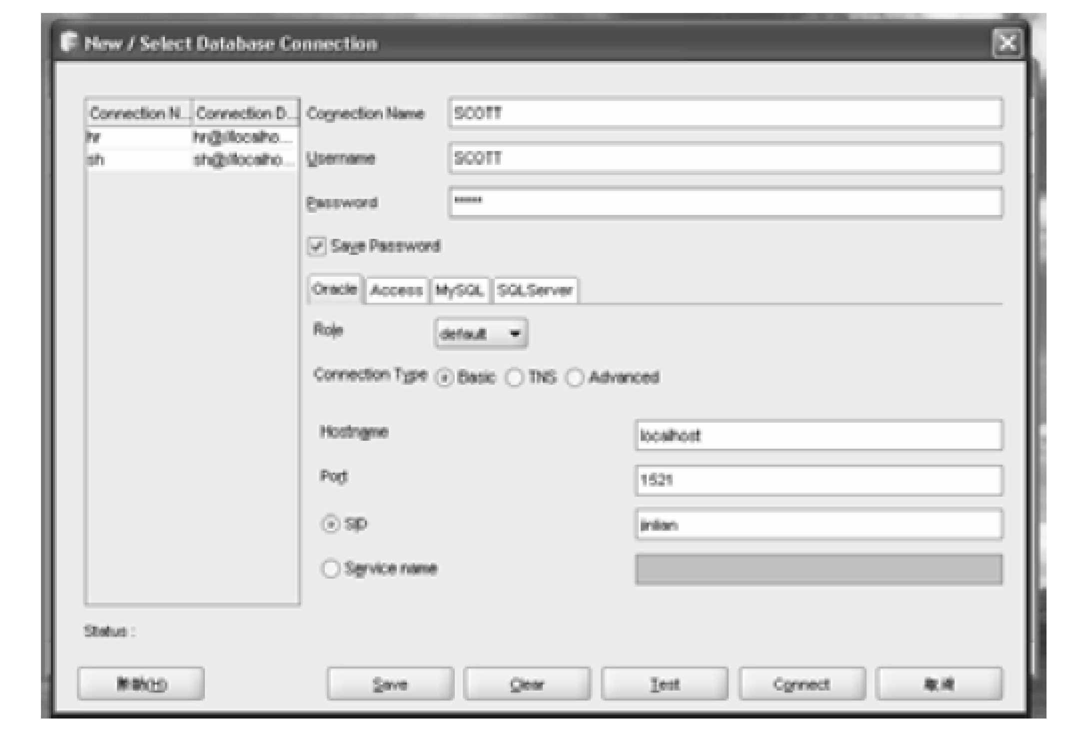


图　17.26

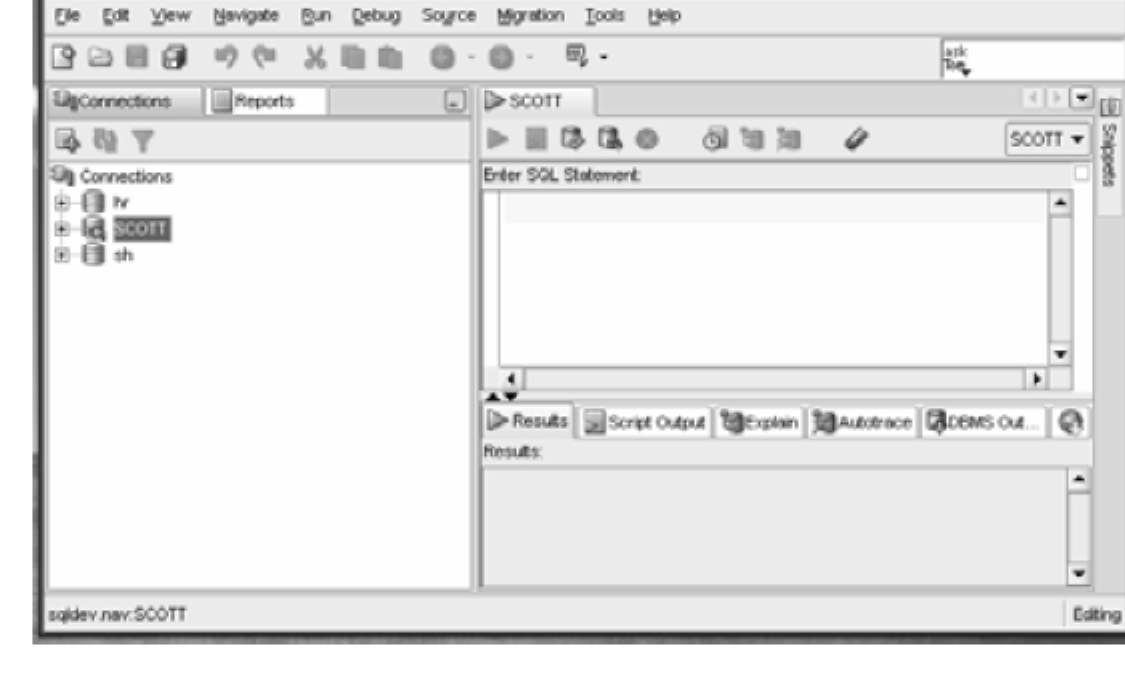


图　17.27

为了导出 emp 表的设计信息，在连接中展开 SCOTT 节点（在我的系统上之前已经创建了两个其他的连接 hr 和 sh），然后再展开 Tables 节点，选择 EMP，然后就会得到如图 17.28 所示的画面。

在图 17.28 的画面中可以获得该表的所有列的设置和表的基本结构。为了获取表之间的关系，可以选择 Constraints 选项卡，就可以得到该表所有约束的信息，如图 17.29 所示。

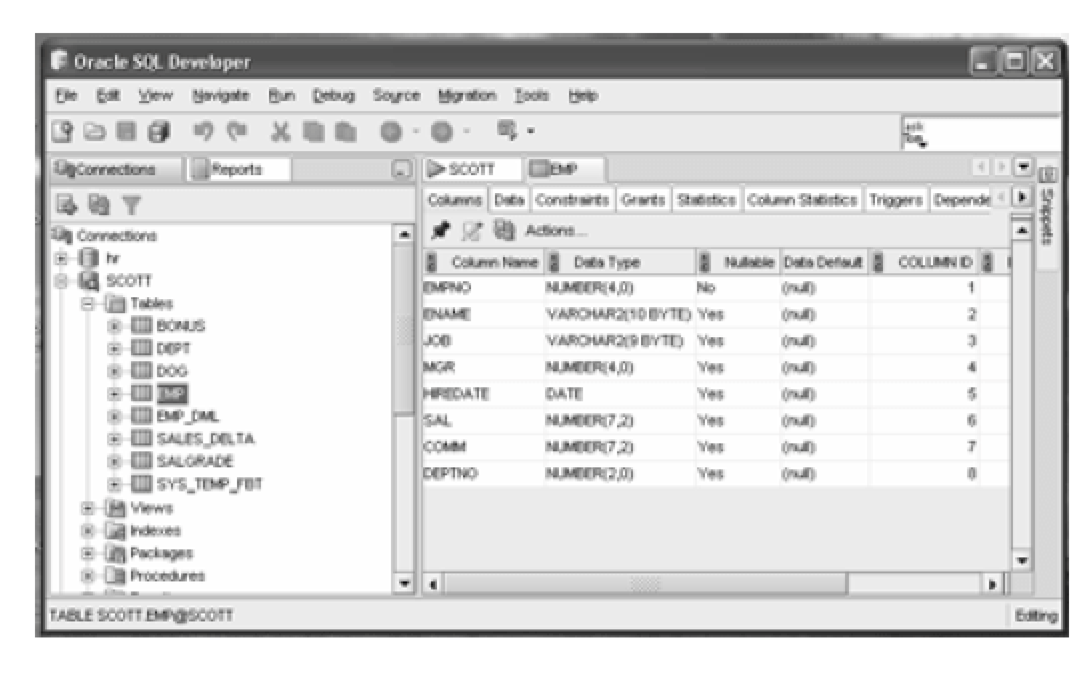


图　17.28

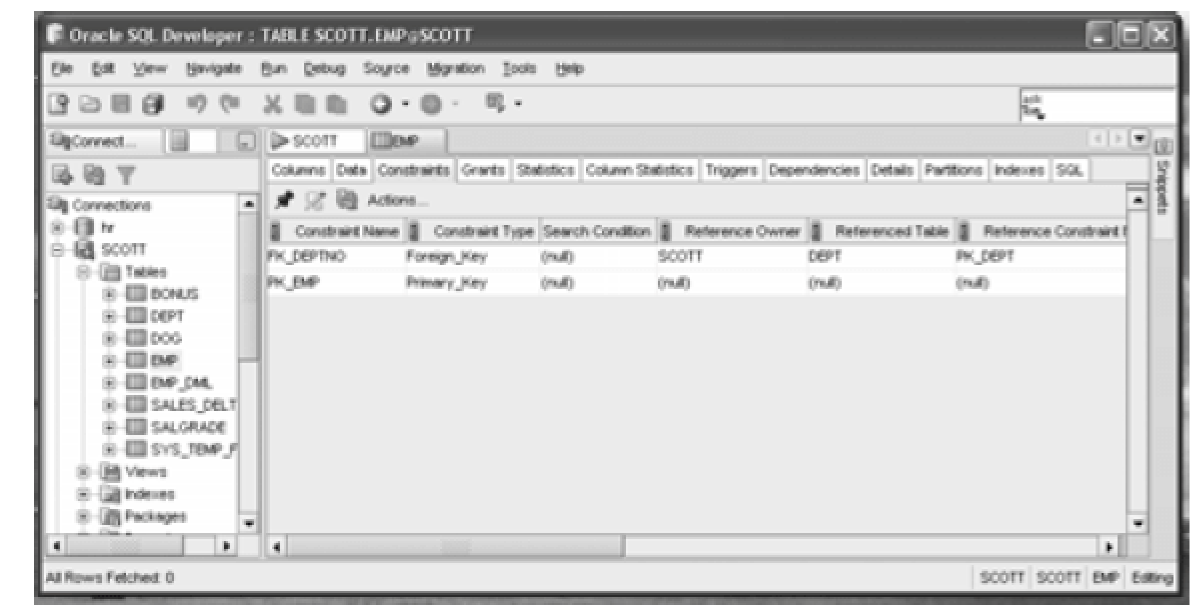


图　17.29

如果对创建 emp 表的 DDL 语句感兴趣，可以选择 SQL 选项卡，就可以得到创建该表所需的 DDL 语句，如图 17.30 所示。也可以将这些 DDL 语句导出到一个脚本文件中。

如果对 emp 表中的索引感兴趣，可以选择 Indexes 选项卡，就可以得到该表上的全部索引信息，如图 17.31 所示。

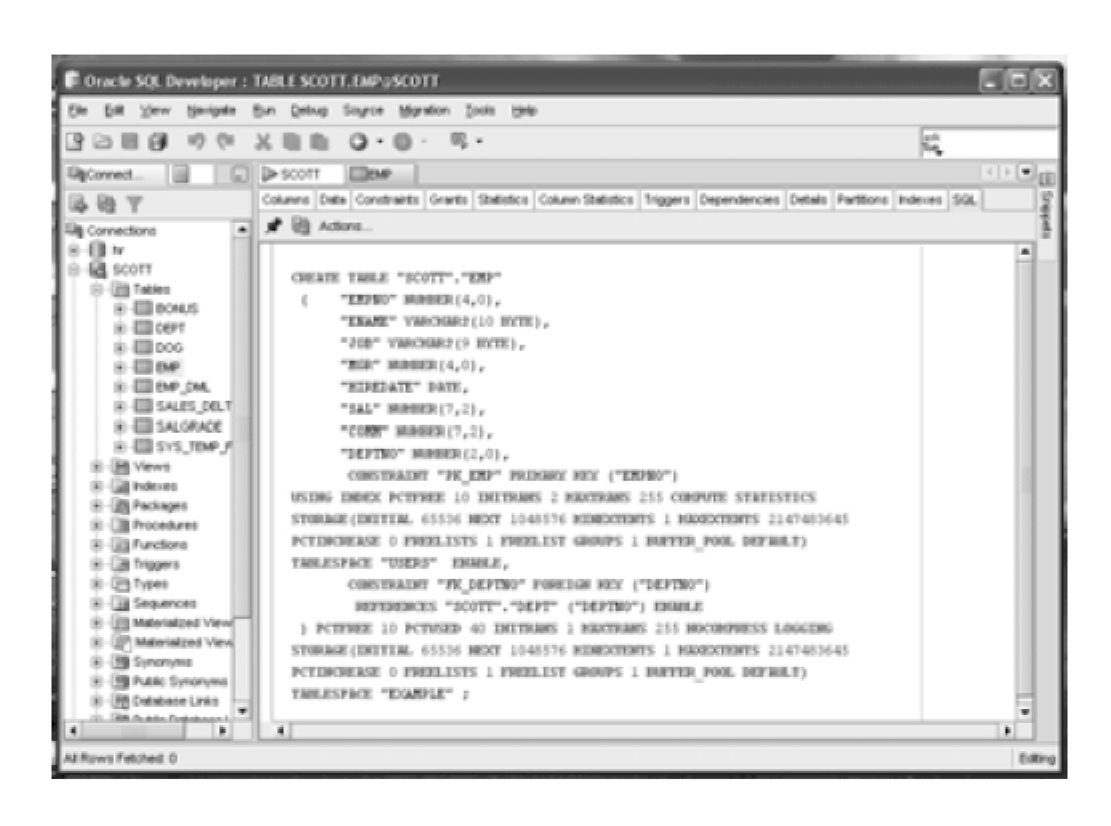


图　17.30

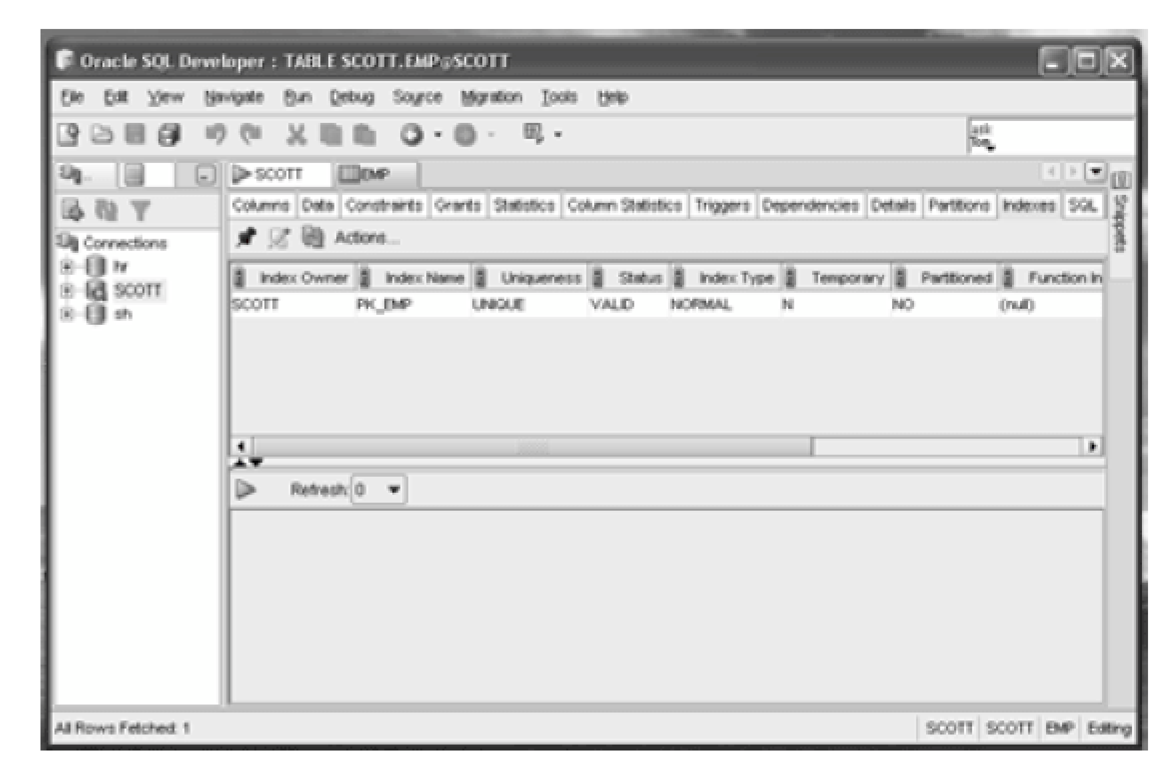


图　17.31

您可以重复以上的操作导出所有表的相关信息，利用这些所获得信息就可以很容易地还原实体-关系图和物理设计。导出了设计之后，我们也可以使用这一工具轻松地导出存储程序的源代码。为此，在连接中选择 hr（如果没有，就要先创建 hr 连接），然后，在 Username 文本框中输入“hr”，在 Password 文本框中输入该用户的密码，最后单击“确定”按钮，如图 17.32 所示。

- 如果要导出存储过程 secure_dml 的源代码，在 hr 连接中选择 Procedures，之后再选择 secure_dml，就会出现如图 17.33 所示的画面。

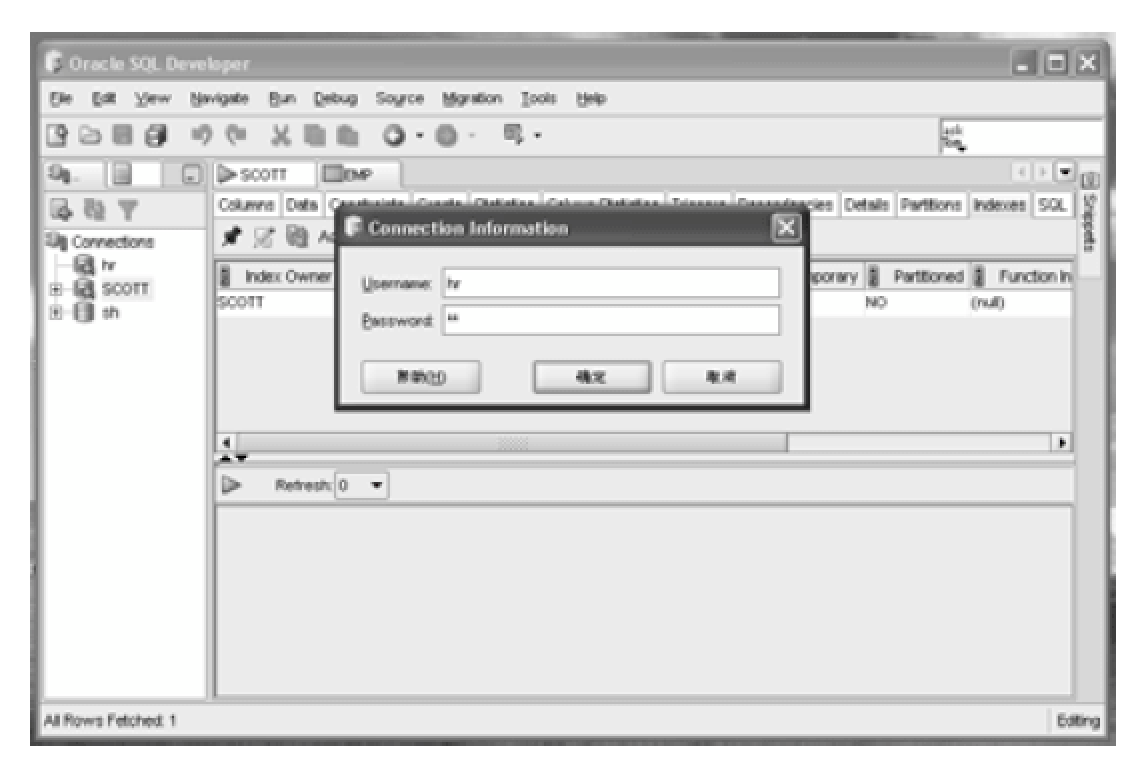


图　17.32

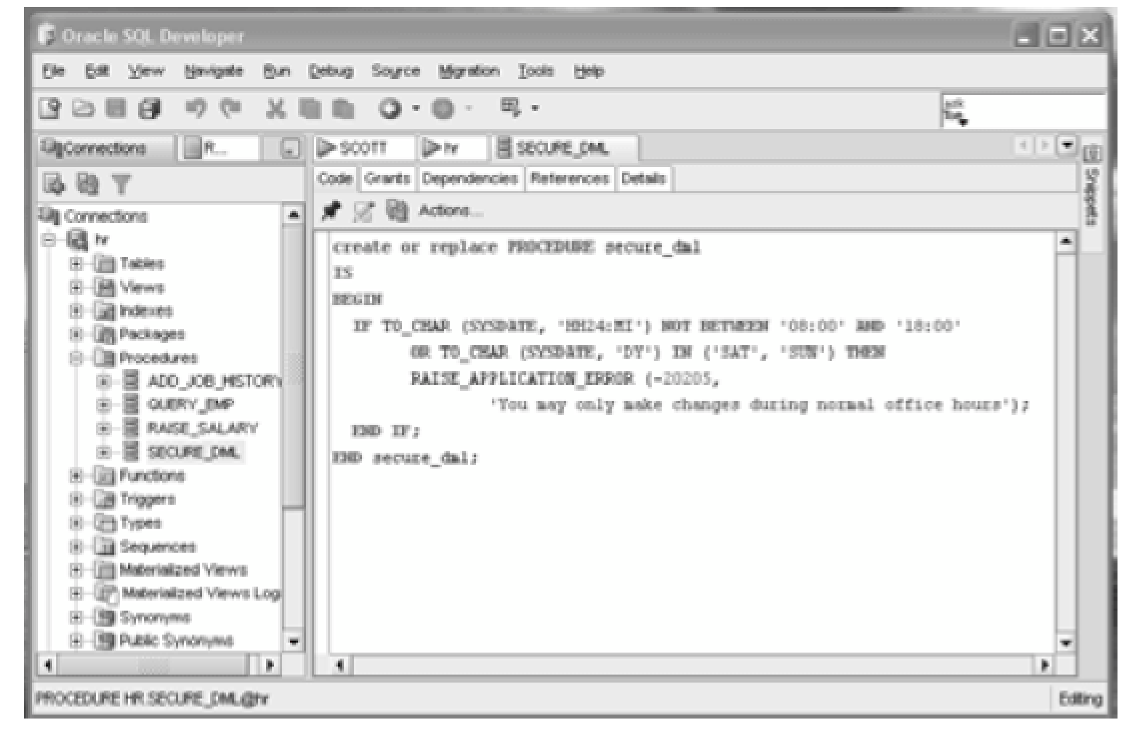


图　17.33

如果觉得这个过程写得不错，可以选择 File→Save As 命令将这些源代码存入一个脚本文件，如图 17.34 所示。

- 如果要导出存储函数 GET_SAL 的源代码，在 hr 连接中选择 Functions，之后再选择 GET_SAL（也可以说其他的存储函数），就会出现如图 17.35 所示的画面。

如果觉得这个函数写得不错，也可以将这个函数的源代码写入一个脚本文件。可以反复使用以上的方法导出所有程序的源代码。其实，人类发展的历史也是这样，几乎所有的帝国在发展和壮大过程中都是靠掠夺和殖民来扩张和昌盛的，因为“偷和抢”总是比“做”快。

接下来我们介绍如何利用 Oracle SQL Developer 导出 SQL 语句的执行计划。为此，选择切换到 SCOTT 连接，使用鼠标右键单击 SCOTT，在弹出的快捷菜单中选择 Open SQL

Worksheet 命令，如图 17.36 所示。

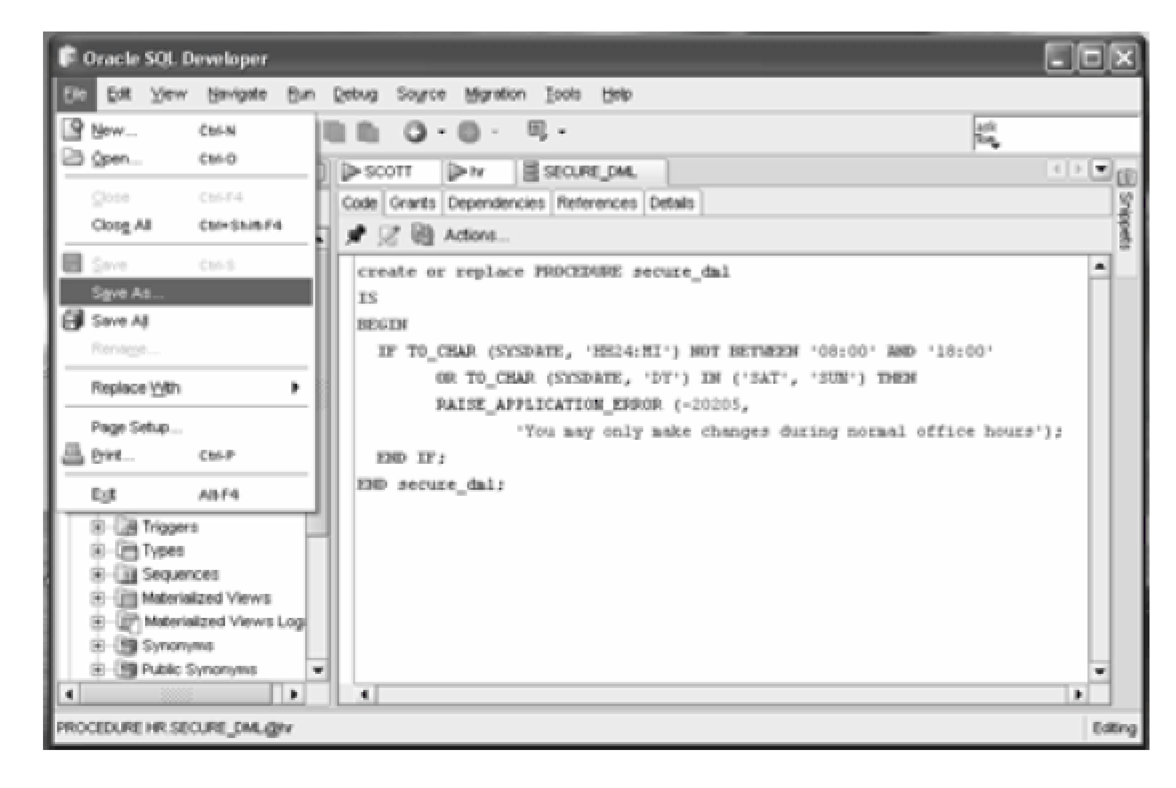


图　17.34

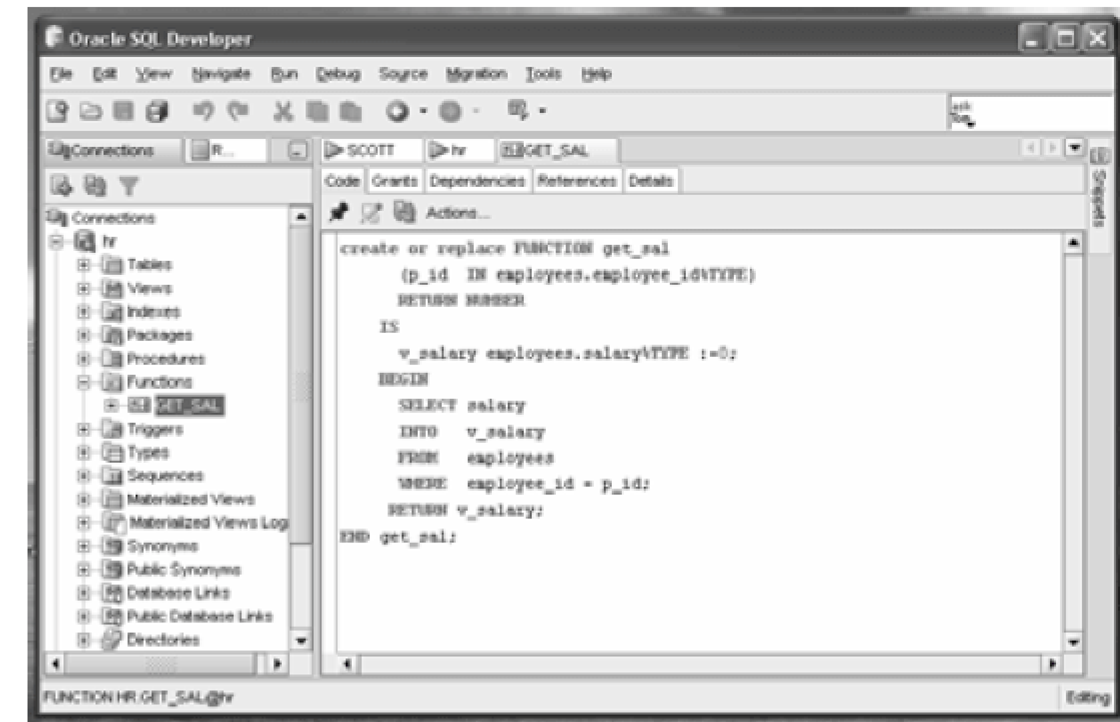
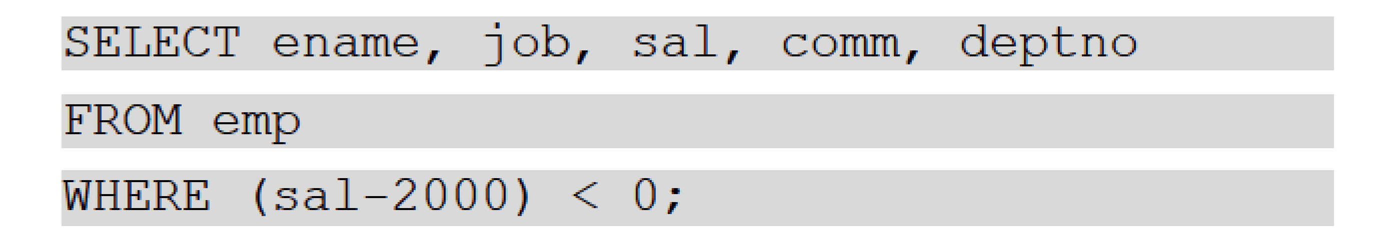


图　17.35

当新的 SQL Worksheet（窗口）打开之后，在里面输入如下的 SQL 查询语句：



```
SELECT ename, job, sal, comm, deptno
FROM emp
WHERE (sal-2000) < 0;
```

我们的目的还是为了检验这个查询是否使用曾经创建的基于表达式 sal-2000 的索引。输入这个 SQL 语句之后，单击 SQL 窗口上面最左边的“执行语句”图标（Execute Statement(F9)），如图 17.37 所示。

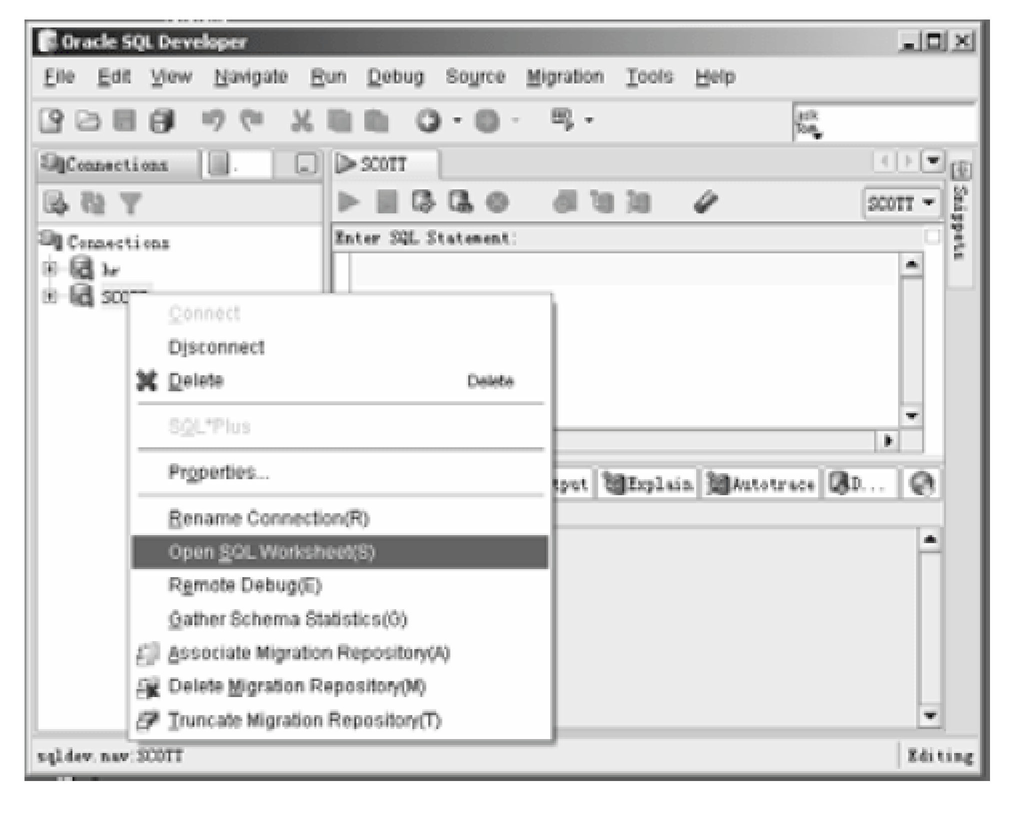


图　17.36

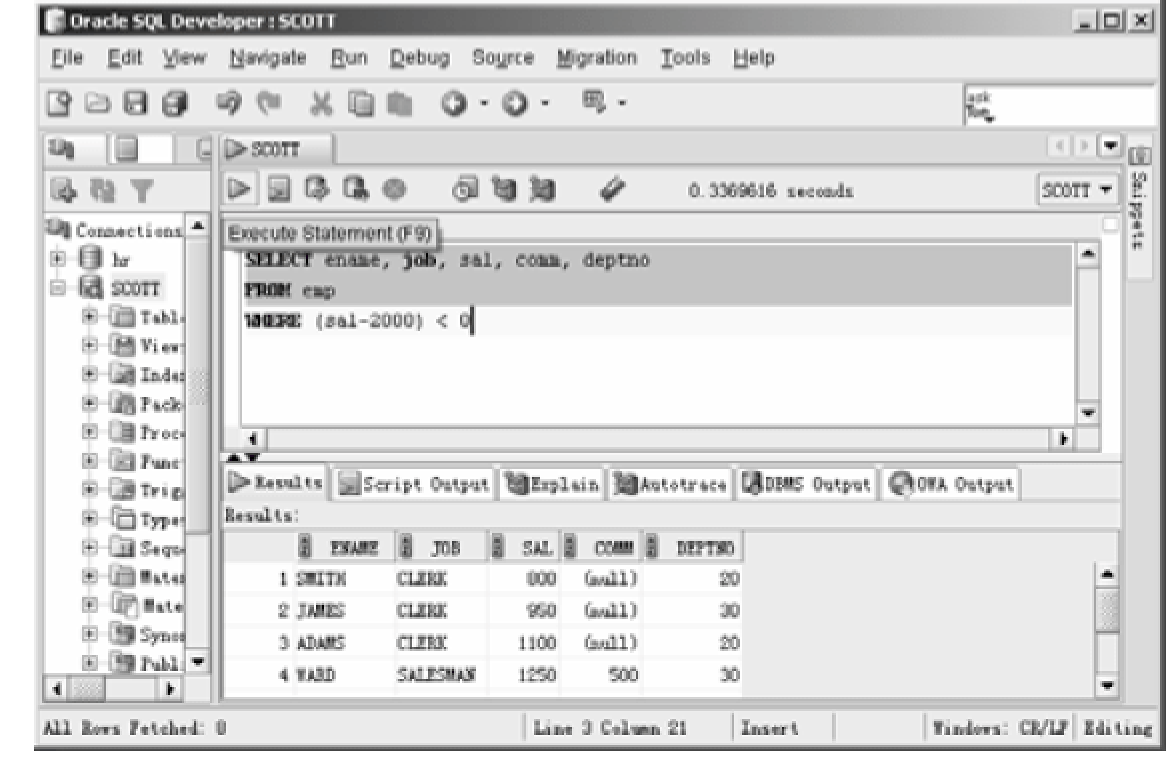


图　17.37

然后就会在 SQL 语句下面的显示窗口中显示查询的结果。此时，单击 SQL 窗口上面右边的第 3 个“执行解释计划”图标（Execute Explain Plan(F6)），如图 17.38 所示。

接着就会在 SQL 语句下面的显示窗口中显示这个查询语句的执行计划，如果没有显示执行计划，只要选择显示窗口上面的 Explain（解释）选项卡就可以显示，如图 17.39　　所示。

除了以上所介绍的功能之外，Oracle SQL Developer 还包括了许多程序开发和调试的功能。如果您想快速地了解 Oracle SQL Developer 基本使用，可以选择 Help→Table of Contents 命令，如图 17.40 所示。

然后，单击 Tutorial: Creating Objects for a Small Database 超链接，就会出现如图 17.41 所示的画面。为了显示清晰，我们已经对画面进行了适当的调整。

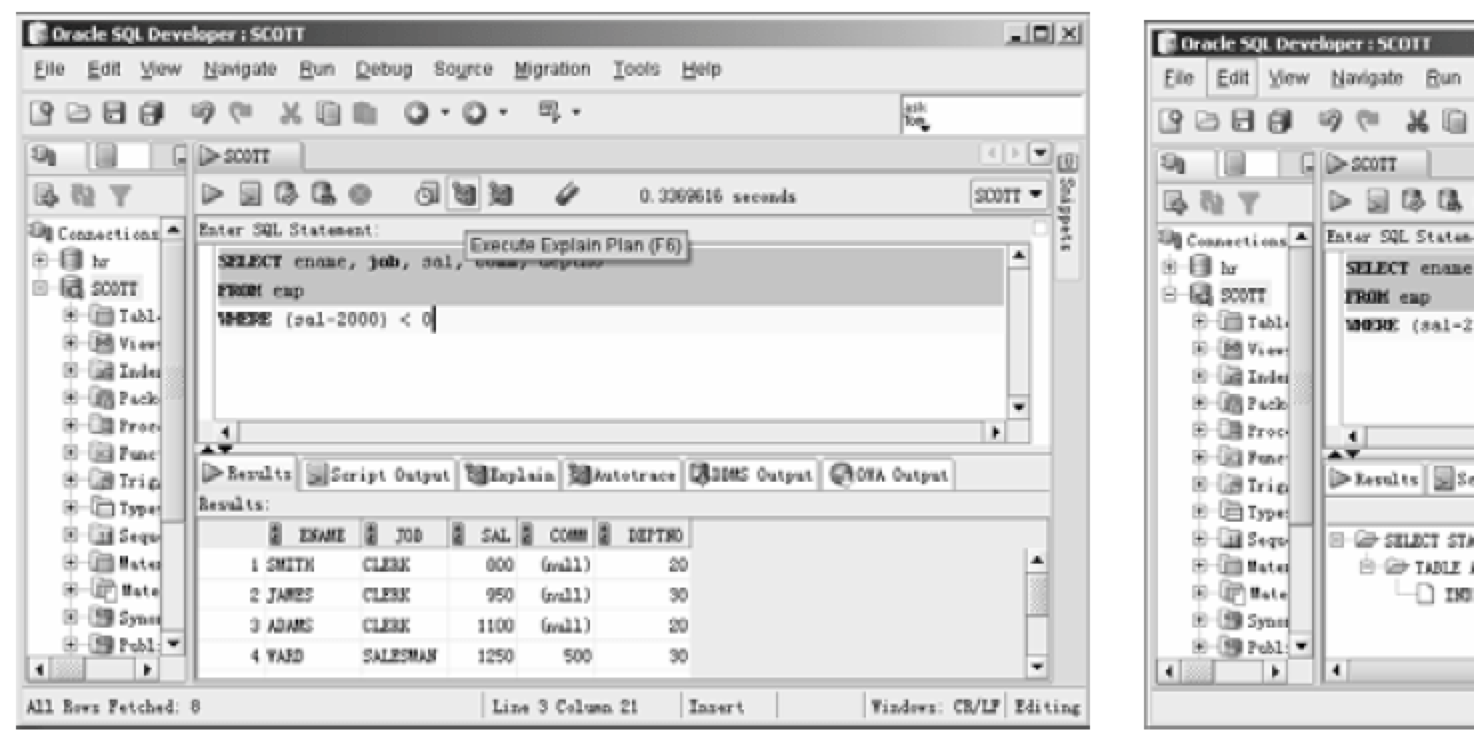


图　17.38

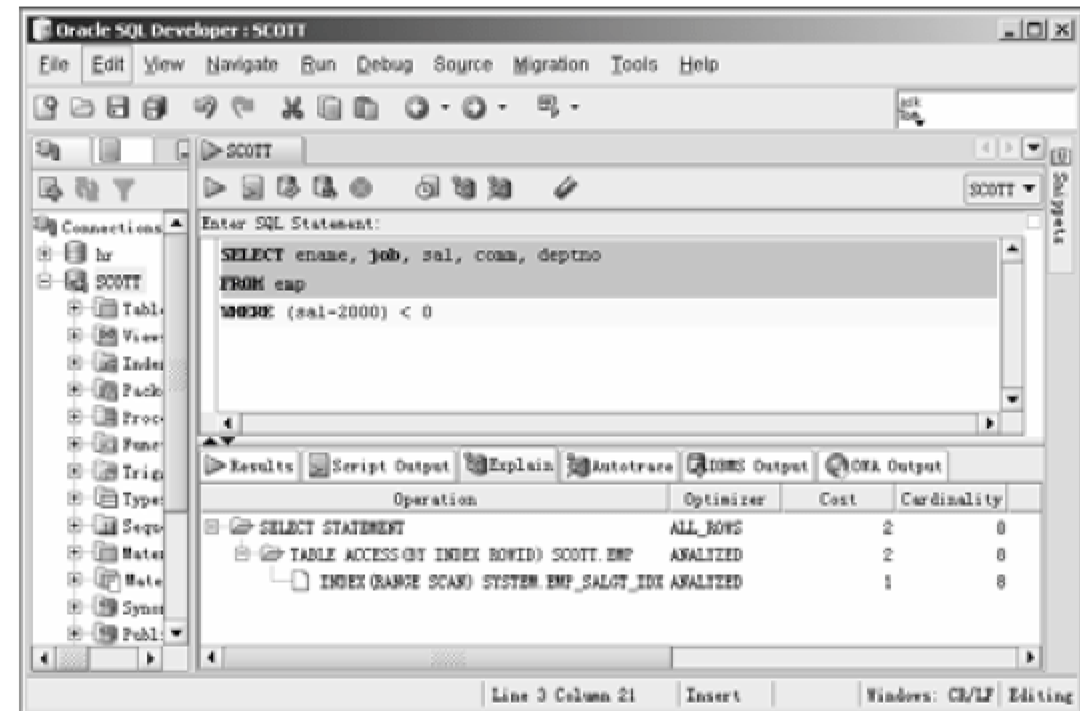


图　17.39

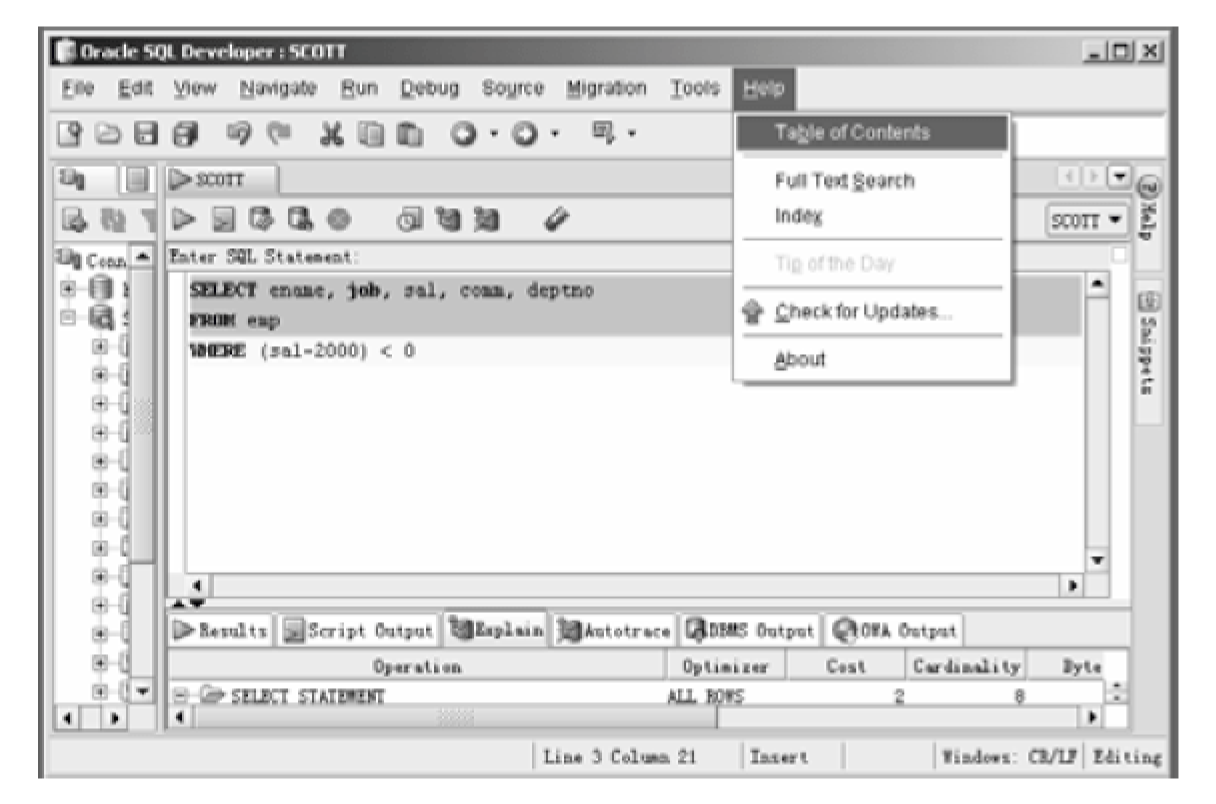


图　17.40

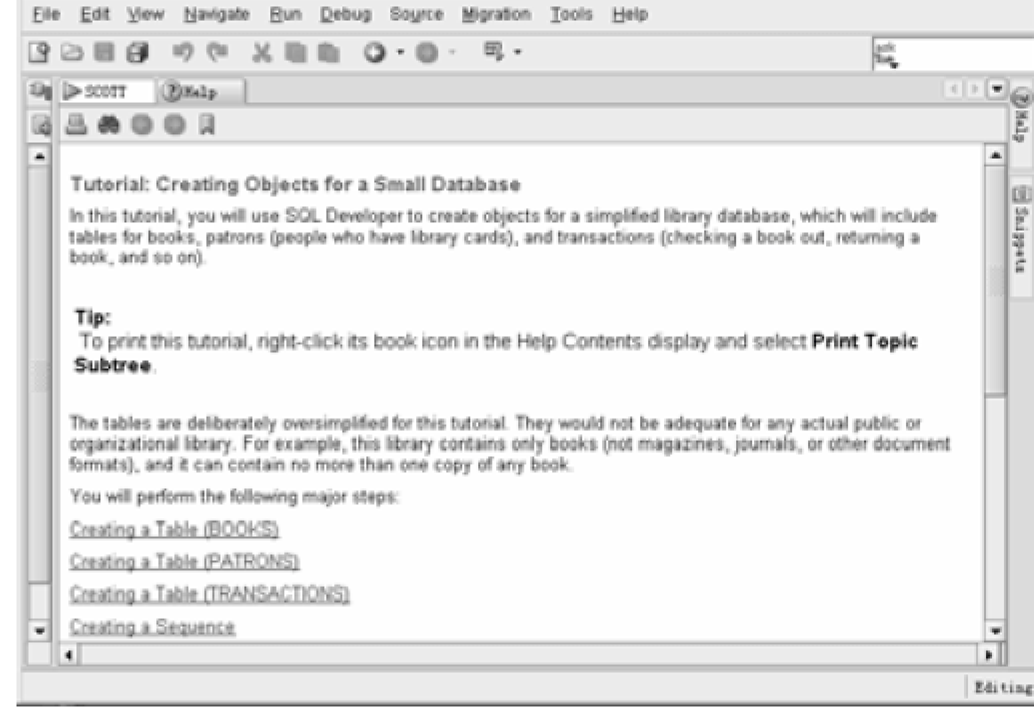


图　17.41

最后，当所有的操作都结束时，就可以选择 File→Exit 命令退出 Oracle SQL Developer，如图 17.42 所示。

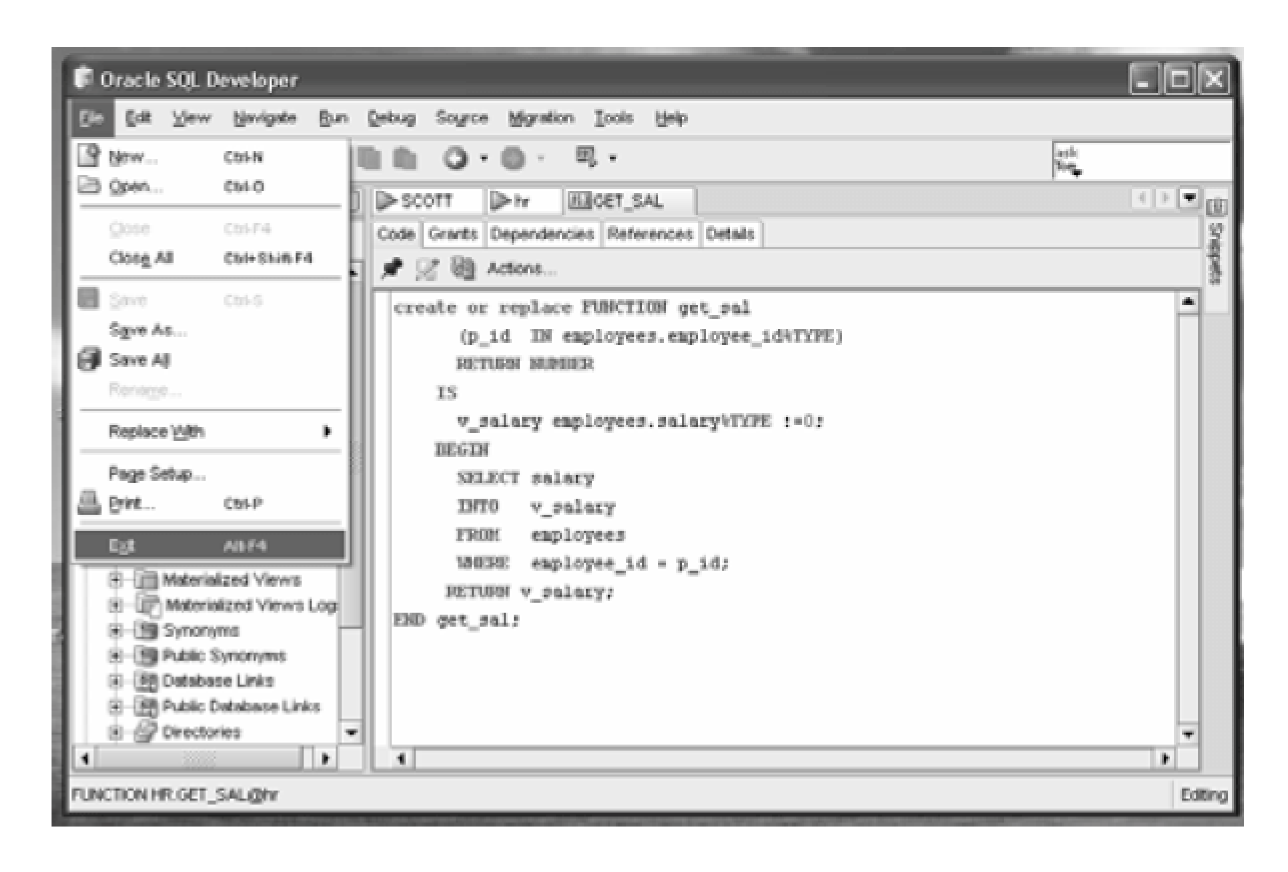


图　17.42

如果读者将来从事的是开发工作或是在一个系统上长期的工作，学会使用一两种图形工具将是十分有益的，因为图形工具会使您的工作更轻松、更快捷，有时也会使开发出来

的程序更稳定。但是作为 Oracle 的从业人员，尤其是数据库管理员，您必须能熟练地使用命令行工具，特别是在系统出问题时，命令行工具很可能是可以救活您的数据库系统的“最后一根稻草”。

17.3 为Oracle SQL Developer配置连接

为了让读者熟悉 Oracle SQL Developer 这一功能强大的（至少 Oracle 公司这么认为）图形开发工具，在这一章中以后的操作都将使用这一工具，为了使读者能够熟悉这一工具在不同版本的数据库中的使用，下面使用的是 Oracle 11g 数据库（Oracle 11g 默认已经安装了 Oracle SQL Developer）。首先要配置连接，具体操作步骤如下：

（1）选择“开始”→“所有程序”→Oracle-OraDb11g_home1→“应用程序开发”→SQL Developer 命令，启动 Oracle SQL Developer，如图 17.43 所示。



图 17.43

☞ 指点迷津：

为了以后操作方便，在出现图 17.43 时，您可以按下 Ctrl 键，用鼠标将 SQL Developer 图标拖到桌面上。

（2）如果是第一次启动 SQL Developer，可能要求您输入 java.exe 的全路径。此时，启动资源浏览器，进入 Oracle 的安装目录，如图 17.44 所示。

（3）单击“搜索”图标，弹出“搜索助理”窗格，在“全部或部分文件名：”文本框中输入 java.exe、在“在这里寻找：”下拉列表框中选择 app 目录，单击“搜索”按钮，如图 17.45 所示。输入所找到的目录和文件名 F:\app\Administrator\product\11.1.0\db_1\jdk\bin\java.exe 即可完成登录。



图　17.44

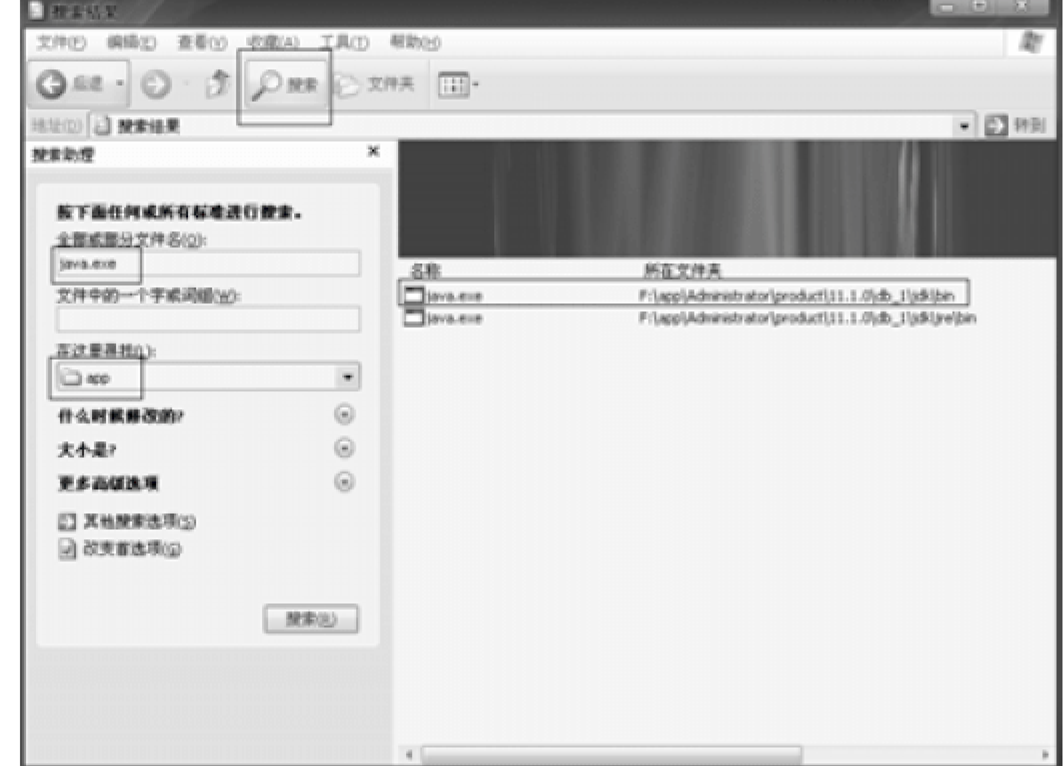


图　17.45

（4）单击 New 图标建立一个新连接，如图 17.46 所示。

（5）在弹出的 New Gallary 对话框中单击 Database Connection 超链接，如图 17.47 所示。

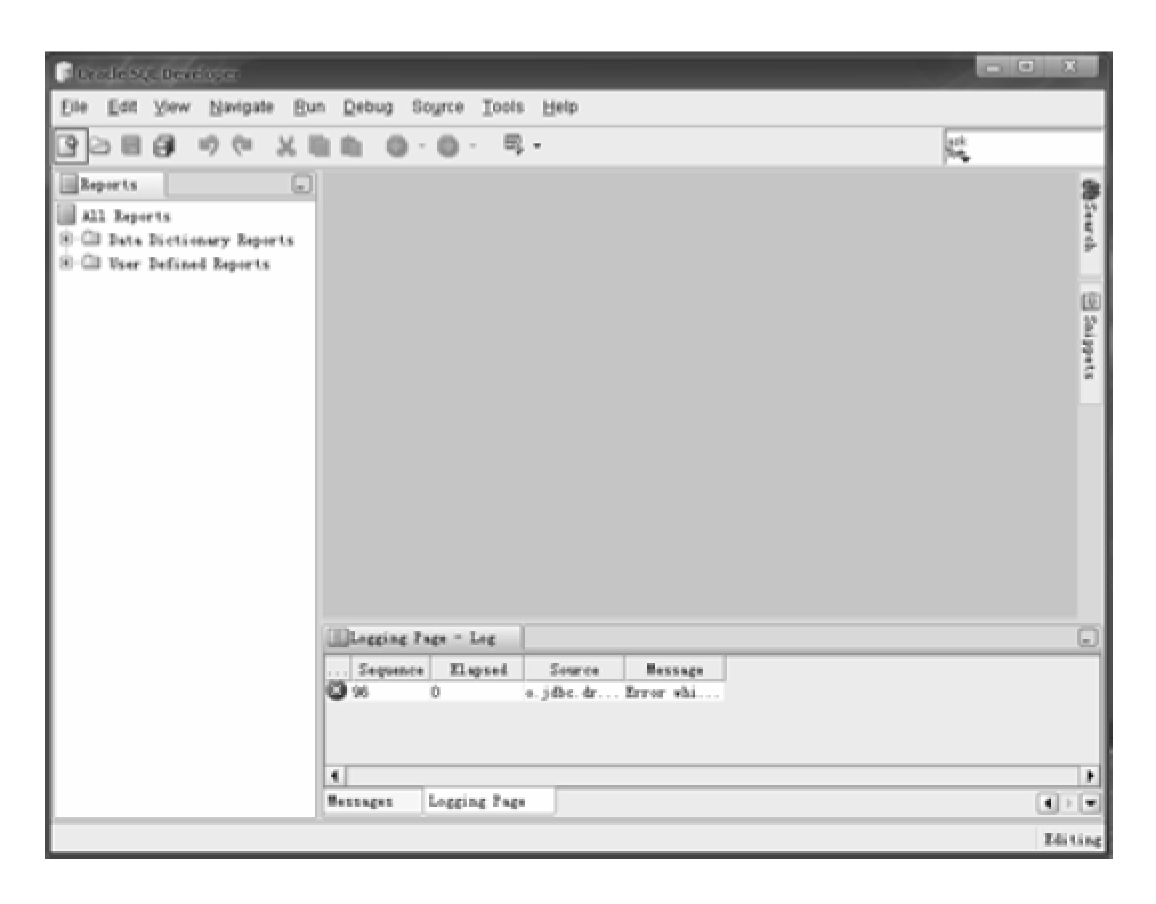


图　17.46



图　17.47

（6）在弹出的对话框的 Connection Name 文本框中输入 HR，在 Username 文本框中也输入 HR，在 Password 文本框中也输入 HR，选中 Save Password 复选框，在 SID 文本框中输入 moon（您的系统上可能不同），其他保持默认设置，单击 Test 按钮，如图 17.48 所示。

（7）如果出现不能建立连接的信息，如图 17.49 所示，可能是因为监听进程（服务）没有启动造成的（为了演示，我在之前故意将监听访问停止了）。此时，您可以选择“开始”→“控制面板”命令，单击“性能和维护”超链接，然后双击“管理工具”图标，最后双击“服务”图标，进入到 Windows 服务窗口。

（8）双击“监听”服务，如图 17.50 所示，将出现监听的属性窗口。

（9）单击“启动”按钮，如图 17.51 所示。

（10）当看到服务状态变为“已启动”之后，单击“确定”按钮，如图 17.52 所示。然后关闭服务窗口。

（11）再次单击 Test 按钮，就会显示连接成功（Success），如图 17.53 所示。

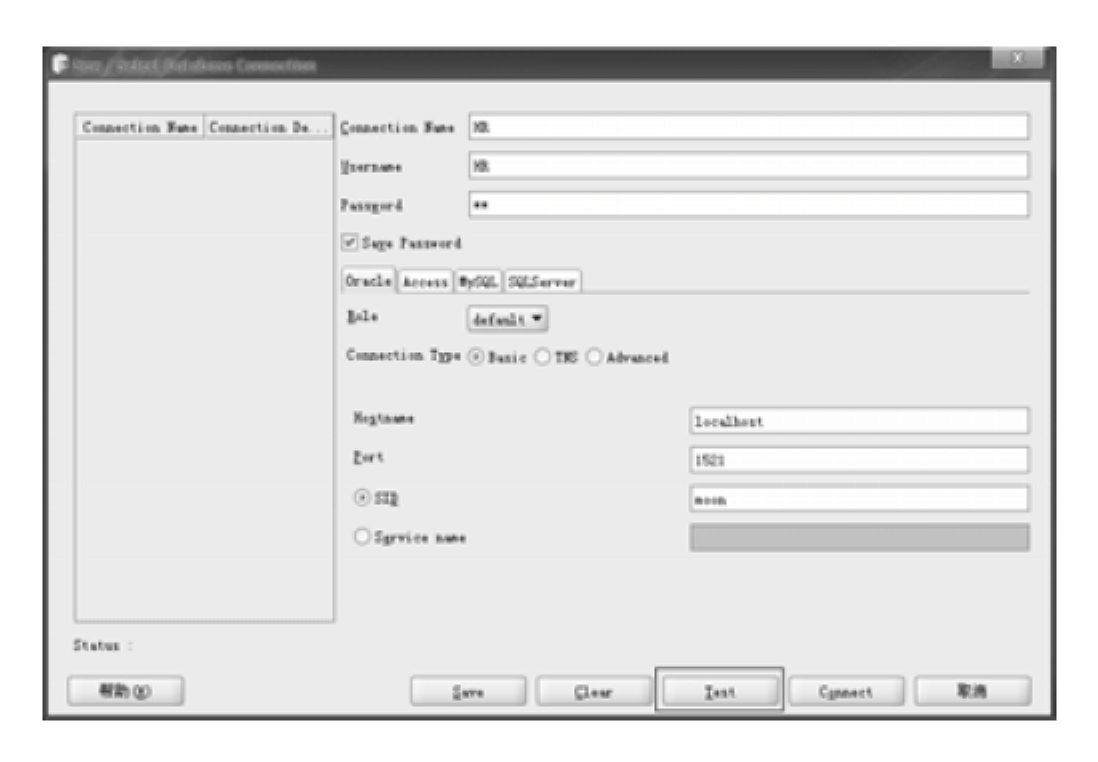


图 17.48

图 17.49

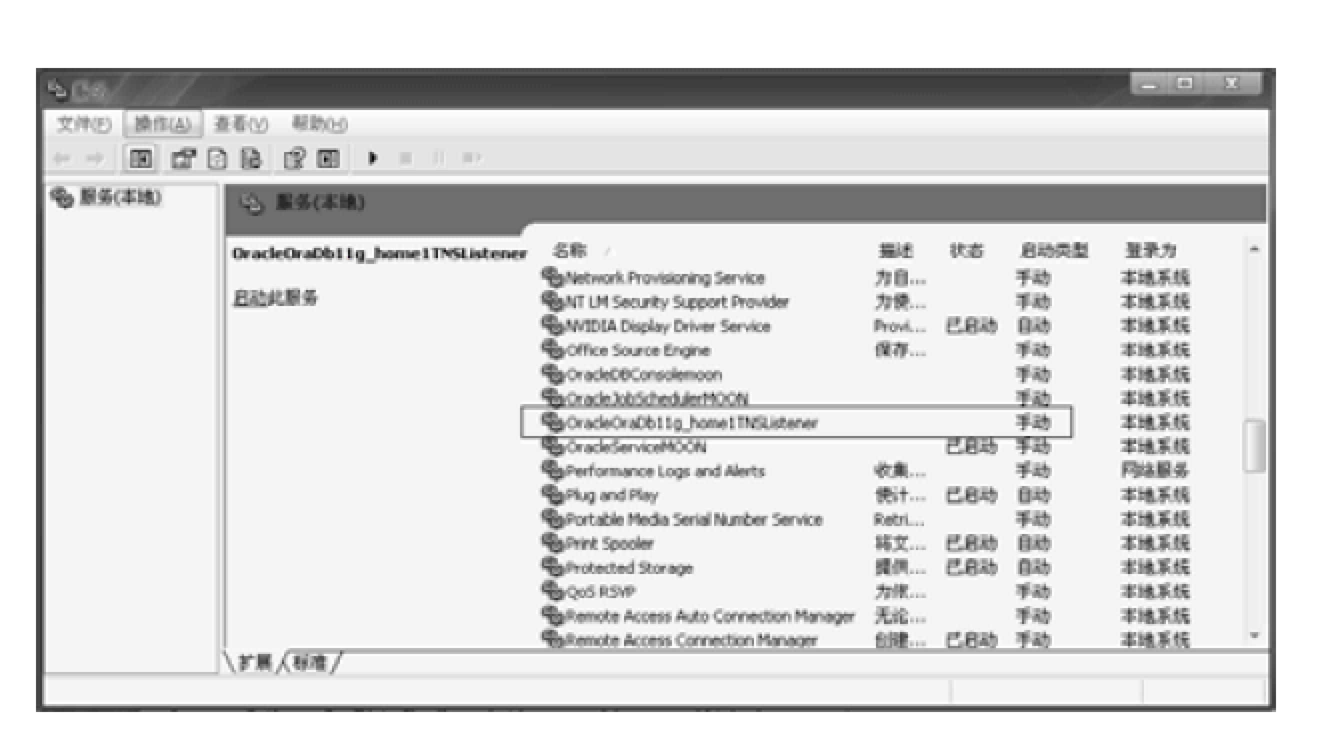


图 17.50



图 17.51



图 17.52

图 17.53

（12）单击 Connect 按钮完成登录，如图 17.54 所示。

（13）出现类似图 17.55 所示的界面就说明登录已经成功。此时可以继续工作了，也可以退出 SQL Developer。选择 File→Exit 命令即可退出这个工具。

可能有读者会觉得每次配置连接太麻烦，其实用不着担心，因为连接只需配置一次，以后就可以反复使用了。

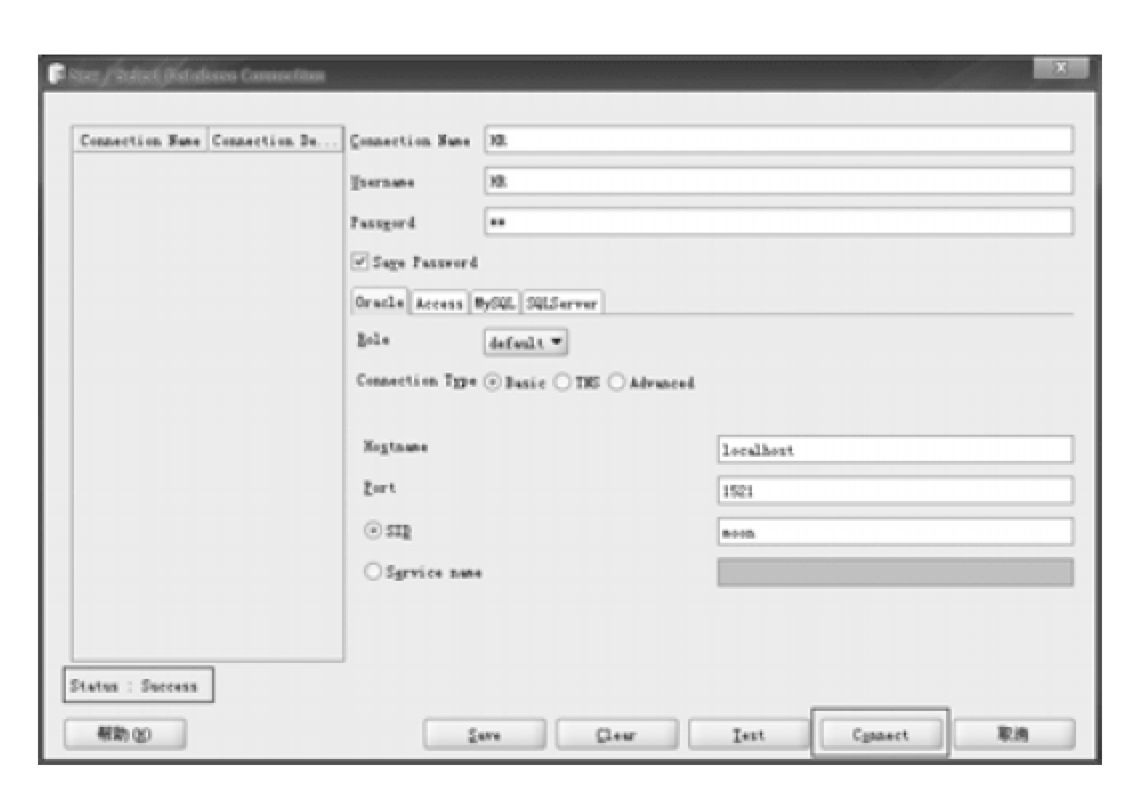


图　17.54

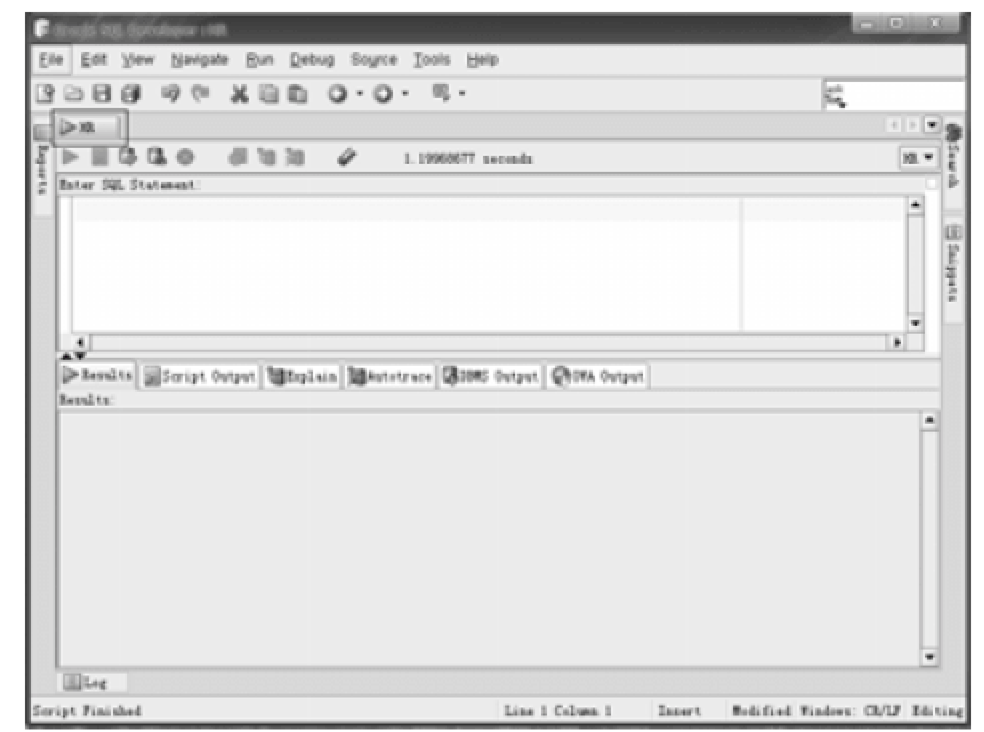


图　17.55

17.4　集合操作符及将使用的表

集合操作（运算）符的功能是将两个或多个查询的结果合并成一个结果。包含集合操作符的查询也称为复合查询。Oracle 一共提供了 4 个集合操作符，如图 17.56 所示。

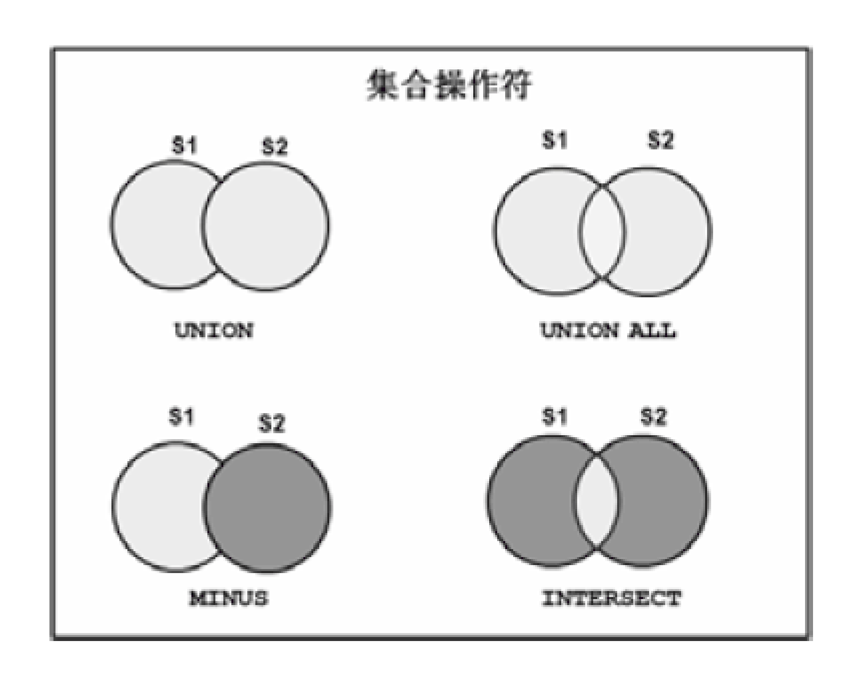


图　17.56

在图 17.56 中，S1 代表第 1 个查询语句的结果，S2 代表第 2 个查询语句的结果，每个简图中的浅色部分代表进行集合操作之后的查询结果。下面简单介绍每个集合操作符的功能。

- UNION：返回两查询的全部结果（数据行），但是要去掉重复行。
- UNION ALL：返回两查询的全部结果（数据行），包括重复行。
- INTERSECT：返回两查询结果的相同数据行。
- MINUS：返回在第 1 个查询结果中但是不在第 2 个查询结果中的数据行。

所有的集合操作符的优先级都是相同的。如果在一个 SQL 语句中包含了多个集合操作符并且没有使用括号，Oracle 服务器解算的次序是从左到右或从上到下。这样的操作优先级规定与我们所熟悉的集合论中的定义有些出入，因为在集合论中 INTERSECT（交集）运算的优先级要高于其他的运算。因此，在进行 INTERSECT 操作时最好使用括号将其括起来，以避免二义性。

因为后面的操作要使用到 SQL Developer，所以要重新启动 SQL Developer。如果在启动后的界面中看不到连接，可以进行如下的操作：

（1）首先选择 View→Connections 命令，如图 17.57 所示将出现所有的连接。

（2）单击 Connections 下的 HR 连接，如图 17.58 所示。

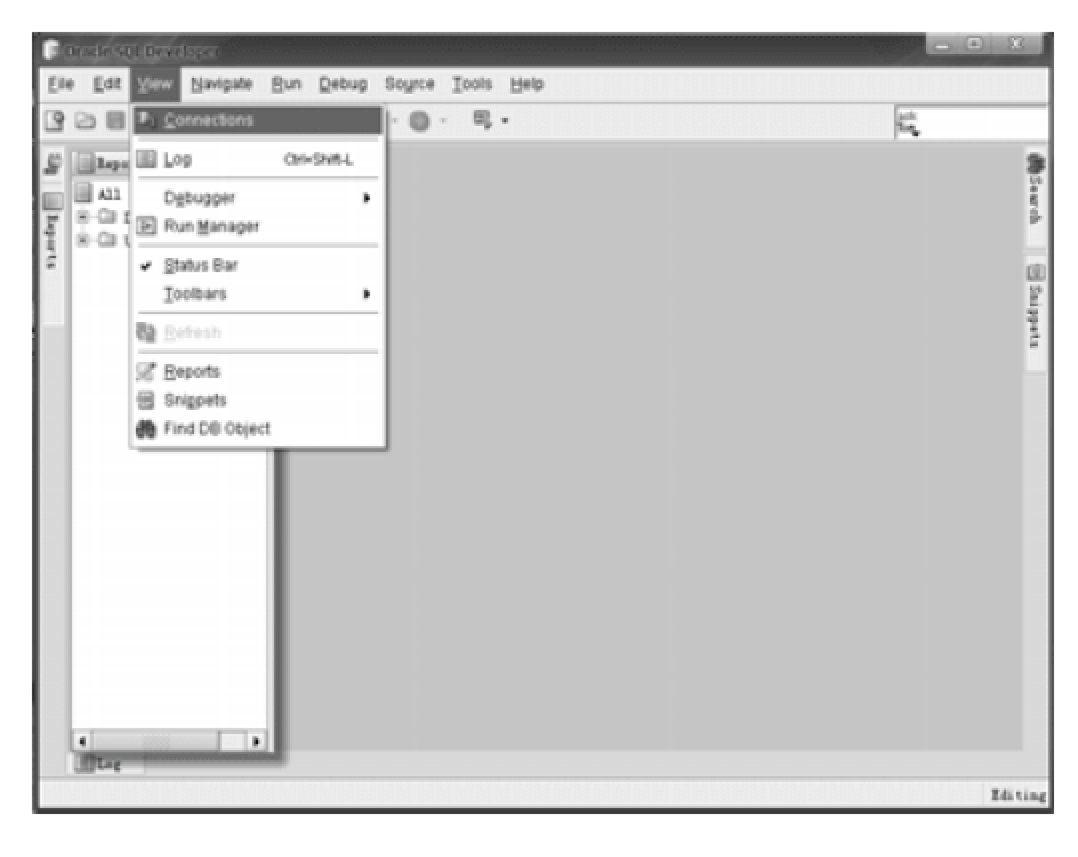


图 17.57

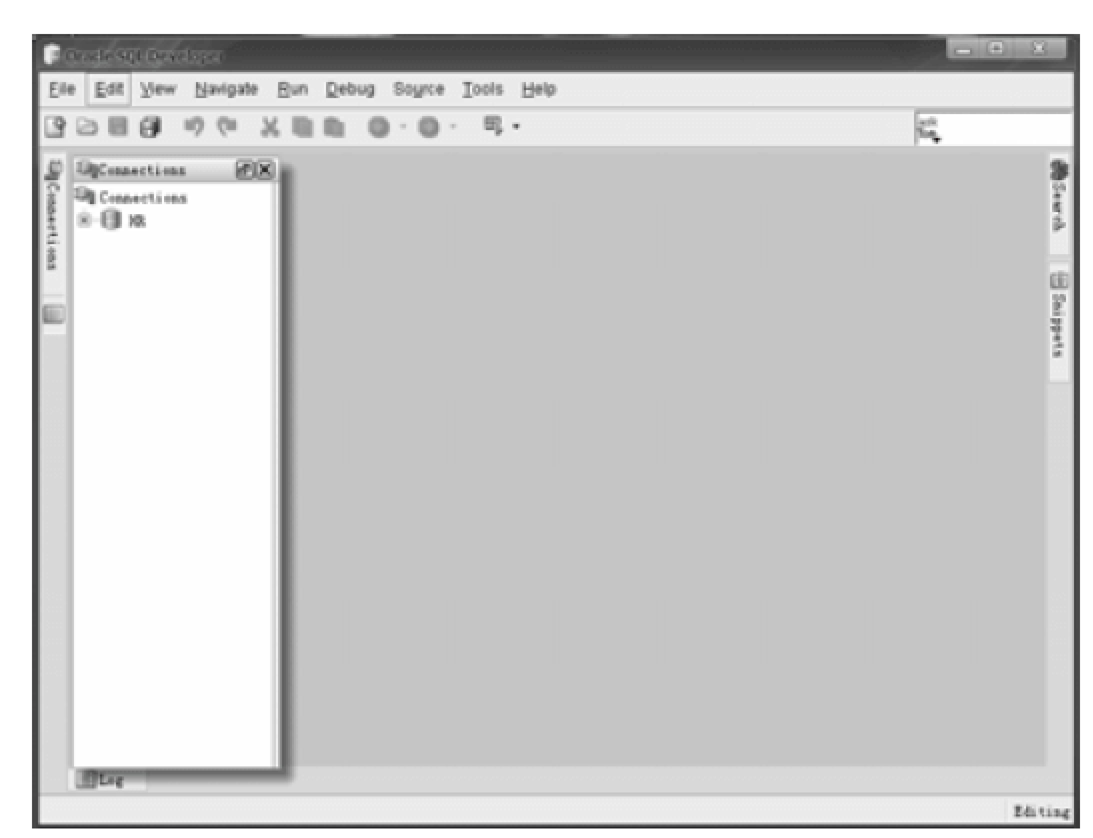


图 17.58

之后将出现如图 17.59 所示的画面，这表示您已经以 HR 用户的身份成功地登录了 Oracle 数据库。

在后面的操作中使用的表共有如下 4 个。

- employees：包括了当前所有员工的详细信息。
- job_history：记录了员工的职位变换的详细历史信息。包括员工在职位发生变化时，前一职位的开始日期、结束日期、职位的标识号和部门。
- departments：包括了所有部门的详细信息。
- locations：包括了每一个部门的地址信息。

可以通过使用 SQL*Plus 的 DESC 命令来查看每个表的结构。例如，为了查看 employees 表的结果，在 SQL Worksheet 区输入 DESC employees 后单击“执行”图标，如图 17.60 所示，将在结果部分显示该表的结构。

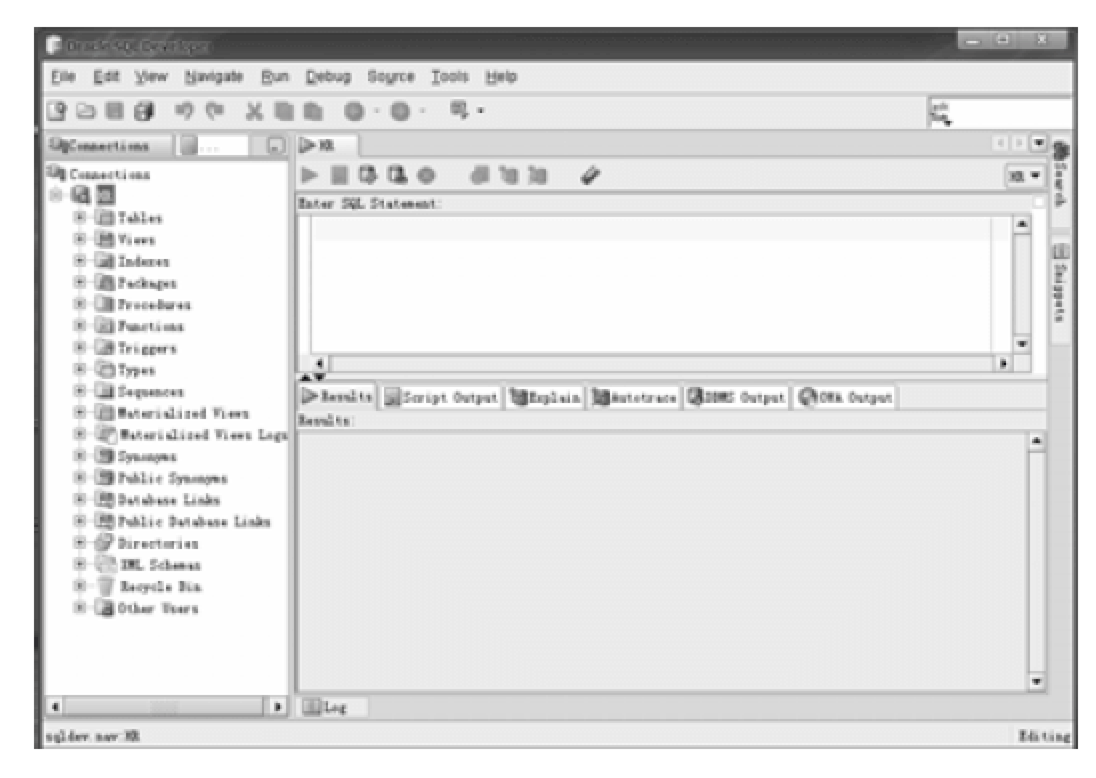


图 17.59

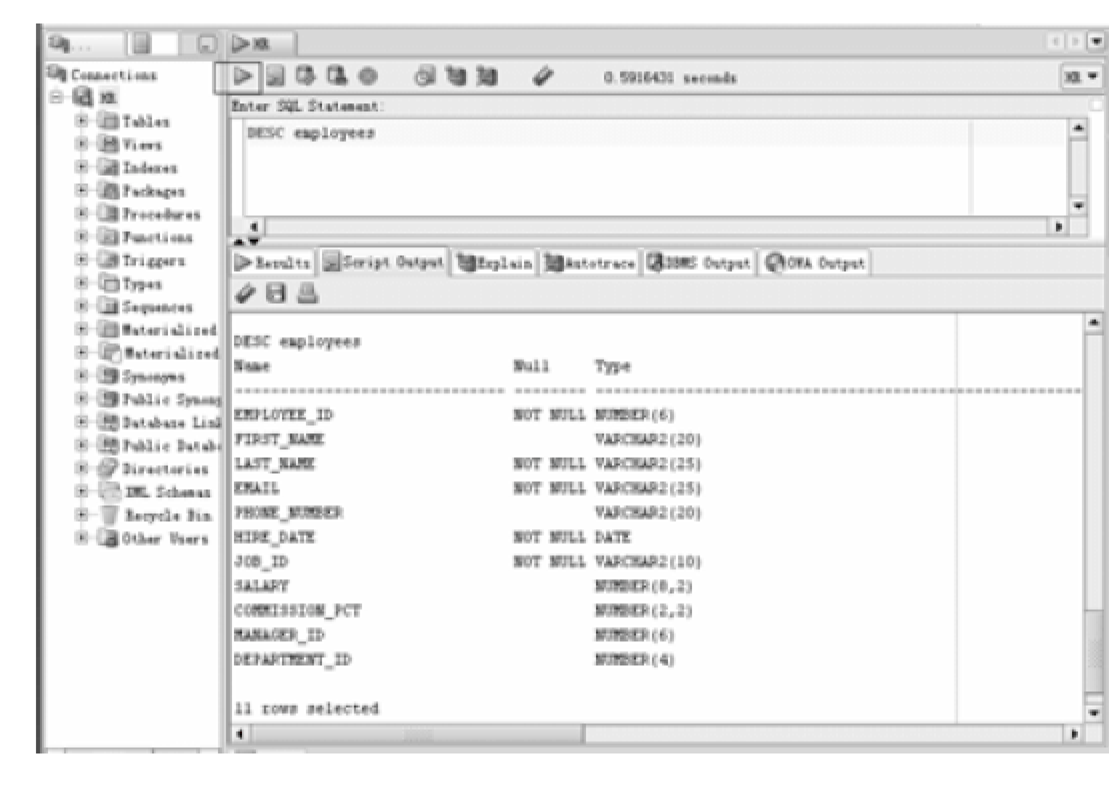


图 17.60

单击“清除”图标就可以清除 SQL Worksheet 区中的命令，如图 17.61 所示。

然后输入类似“select * from employees where rownum <= 15”的 SQL 语句以获取员工的信息，如图 17.62 所示。

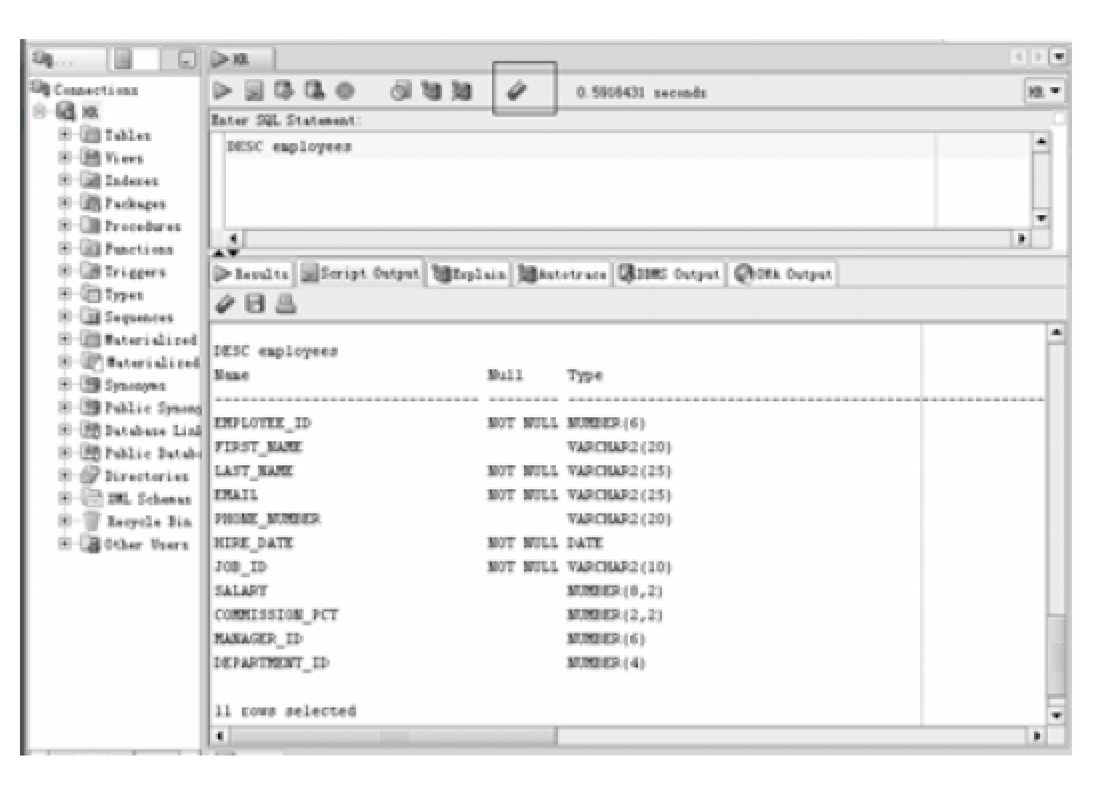


图　17.61

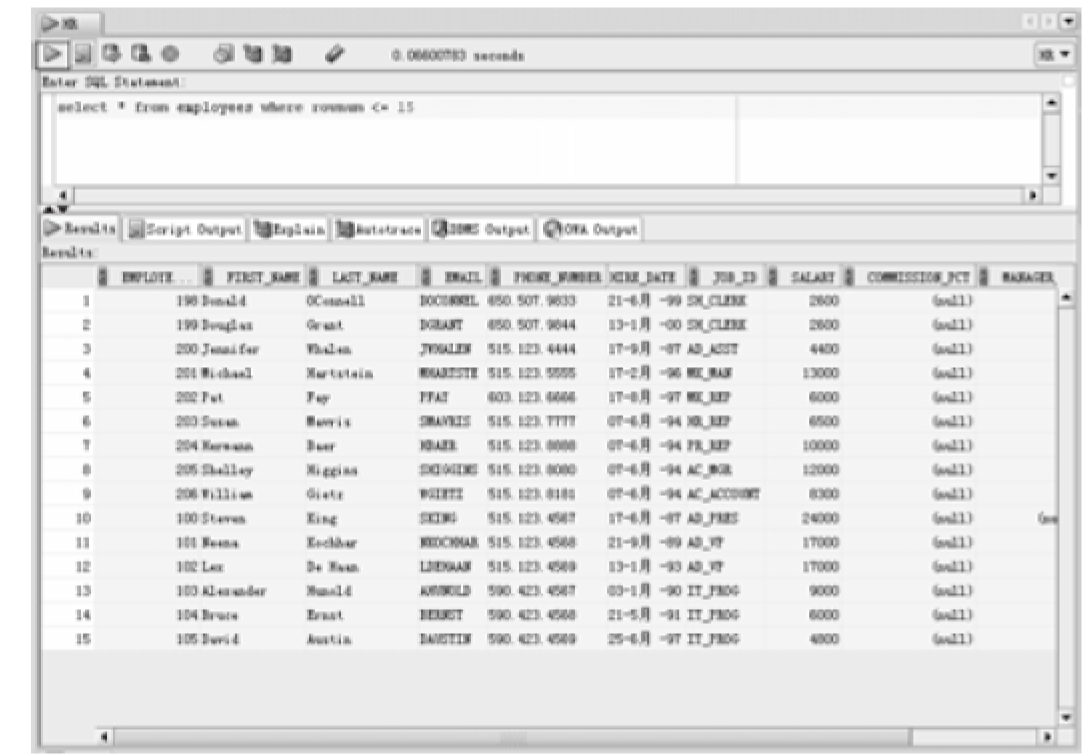


图　17.62

使用类似的方法就可以获取其他 3 个表的结构和所存储的数据的详细信息，有兴趣的读者可以自己尝试一下。在接下来的几节中我们要详细地介绍每种集合操作符的用法。

17.5　UNION集合操作（运算）符

UNION 操作符的功能是返回两个查询的所有结果，但是要去掉重复的数据行。图 17.63 是 UNION 操作符操作结果的示意图。

Oracle 对 UNION 操作做了一些限制，为了使这一操作可以顺利地进行，最好了解如下的规则：

- 每个查询中所选择的列数必须相同。
- 所选择的对应列的数据类型必须属于相同的数据类型组，即数据类型必须匹配。
- 所选择的对应列的列名可以不同。
- UNION 操作符作用在所选择的每一列上。
- 在重复数据行检查期间，不会忽略空值（NULL）。
- 在默认情况下，输出的结果按升序排序。

如果老板想要获取所有员工当前担任的和以前所担任过的职位的详细信息，并要求不包含重复的数据行，那您又该怎样完成他的重托呢？当然一种简单的方法就是使用带有 UNION 运算符的查询语句，如例 17-1。

例 17-1

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

在 SQL Developer 的 SQL Worksheet 区域中输入例 17-1 的 SQL 语句，然后单击 SQL Worksheet 区域左上角的“执行语句”图标（或按 F9 键），如图 17.64 所示。

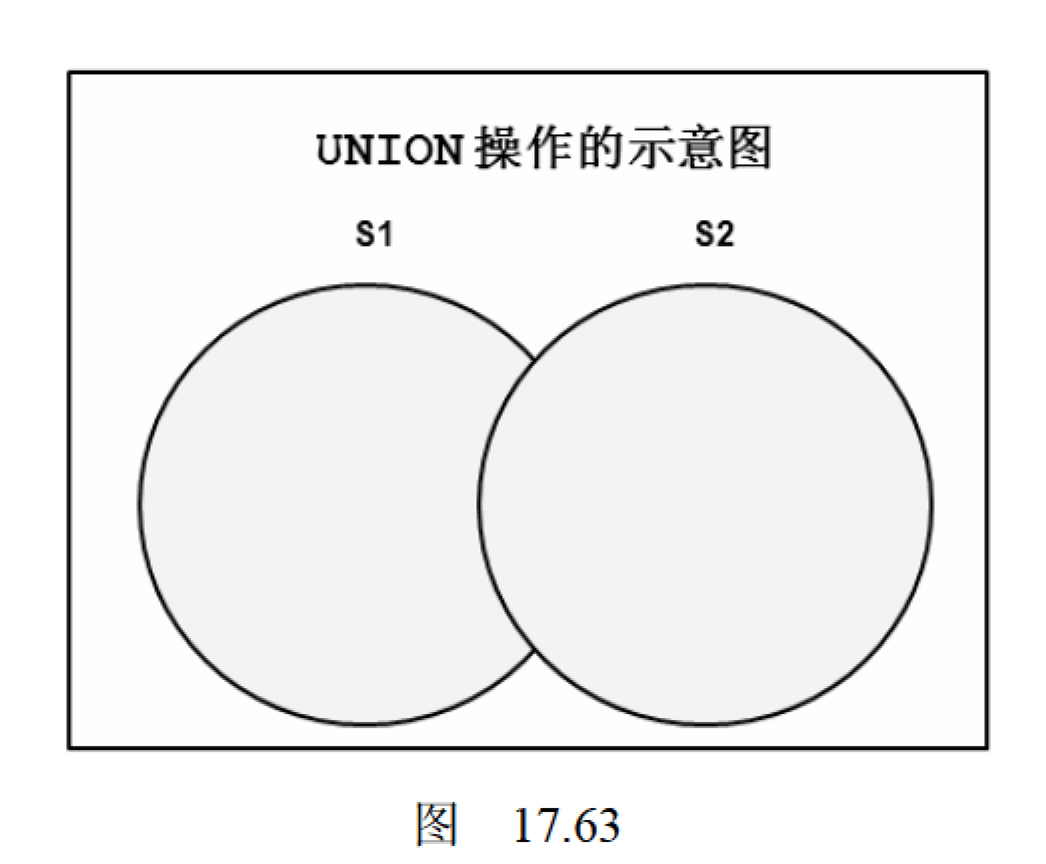


图 17.63

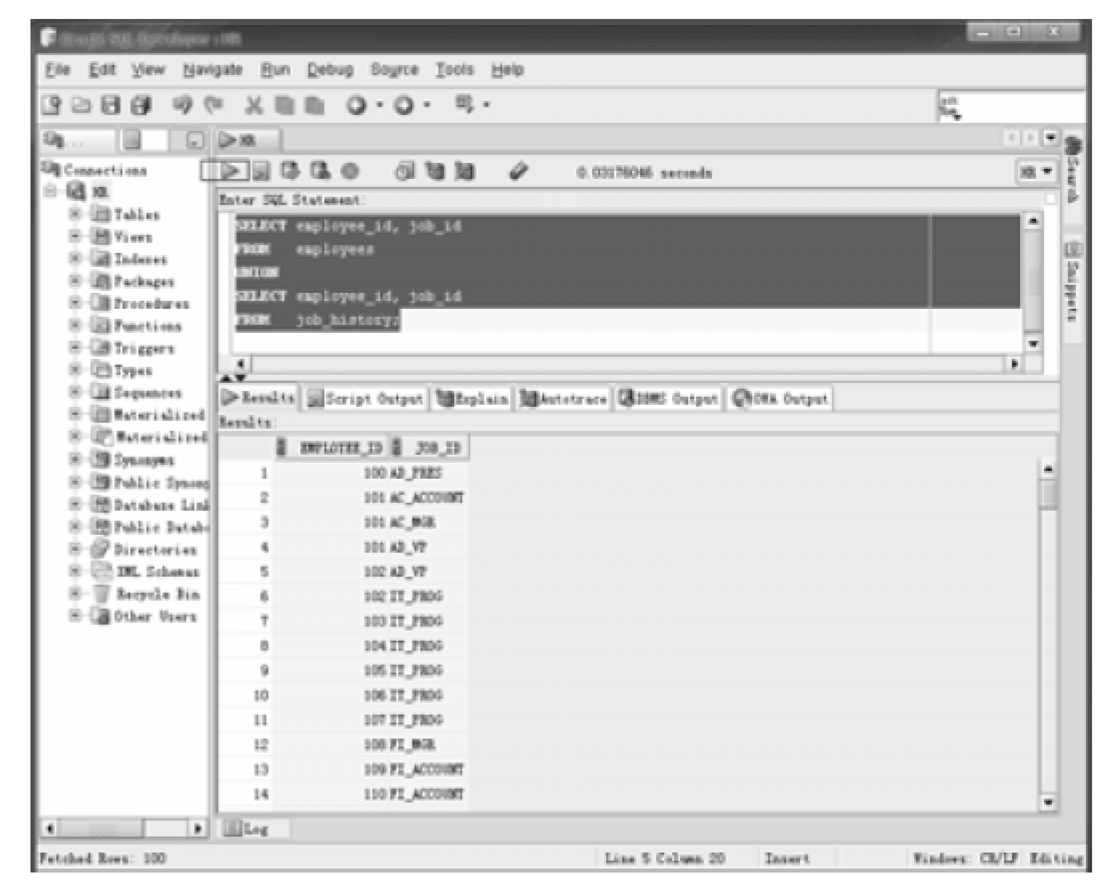


图 17.64

之后就会在 Results 区域得到该语句的运行结果。注意，在每一行的最左边，SQL Developer 都将自动地放上行号。

“执行语句”图标右侧是“运行脚本”图标，如果单击“运行脚本”图标（或按 F5 键）也会得到 SQL 语句的结果，如图 17.65 所示。

以图 17.65 的方式运行 SQL 语句与使用图 17.64 的方式运行 SQL 语句的唯一不同是，它所产生的 SQL 语句结果不包括行号。重新单击“执行语句”图标，然后向下滚动滚动条直到看到第 82 和第 83 行数据为止，如图 17.66 所示。

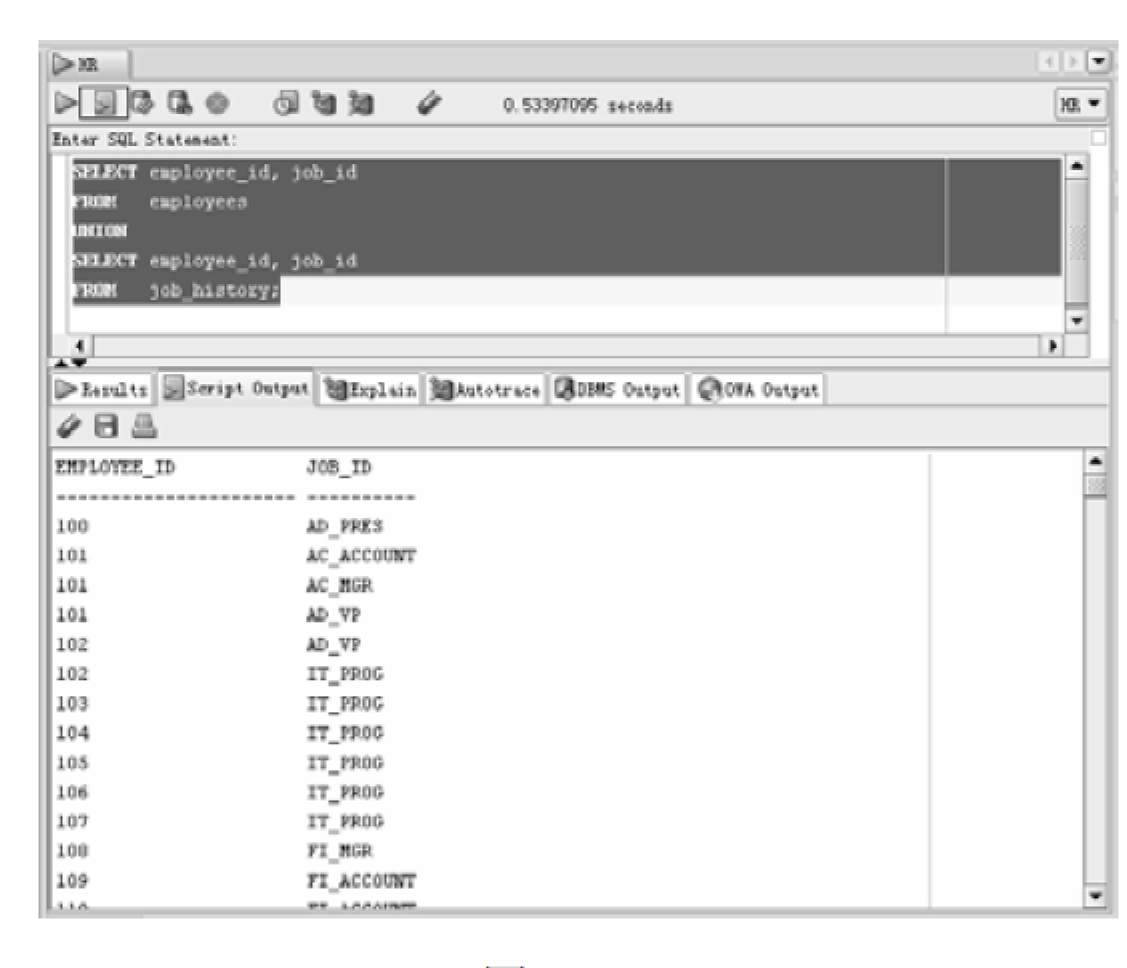


图 17.65

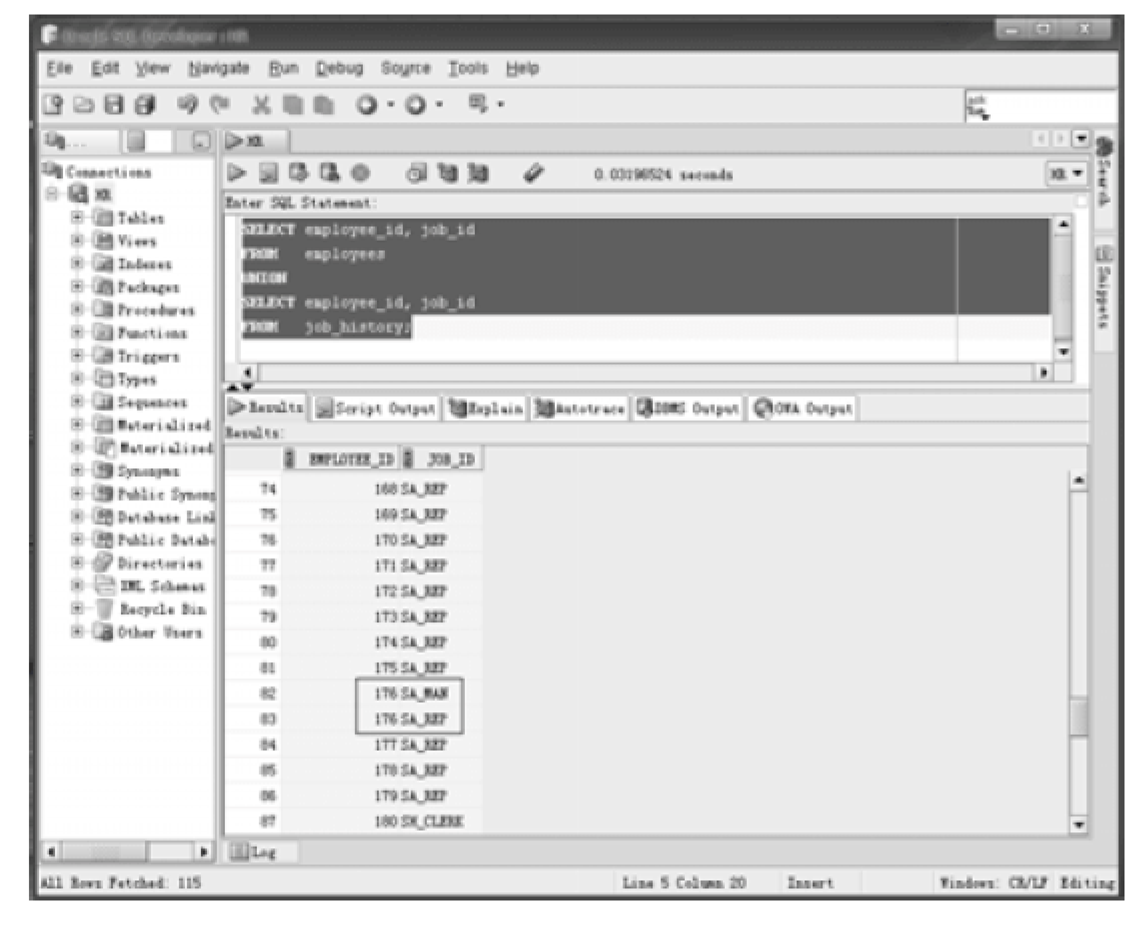


图 17.66

第 82 和第 83 行的数据表明，EMPLOYEE_ID 为 176 的员工担任过两个不同的职位，一个是 SA_MAN（销售经理），一个是 SA_REP（销售代表）。然后继续向下滚动滚动条直到最后，如图 17.67 所示。

第 107 和第 108 行的数据表明，EMPLOYEE_ID 为 200 的员工担任过两个不同的职位，而第 109 和第 110 行的数据表明，EMPLOYEE_ID 为 201 的员工也担任过两个不同的职位。从图 17.67 还可以得知该 SQL 语句执行的结果一共返回 115 行数据。

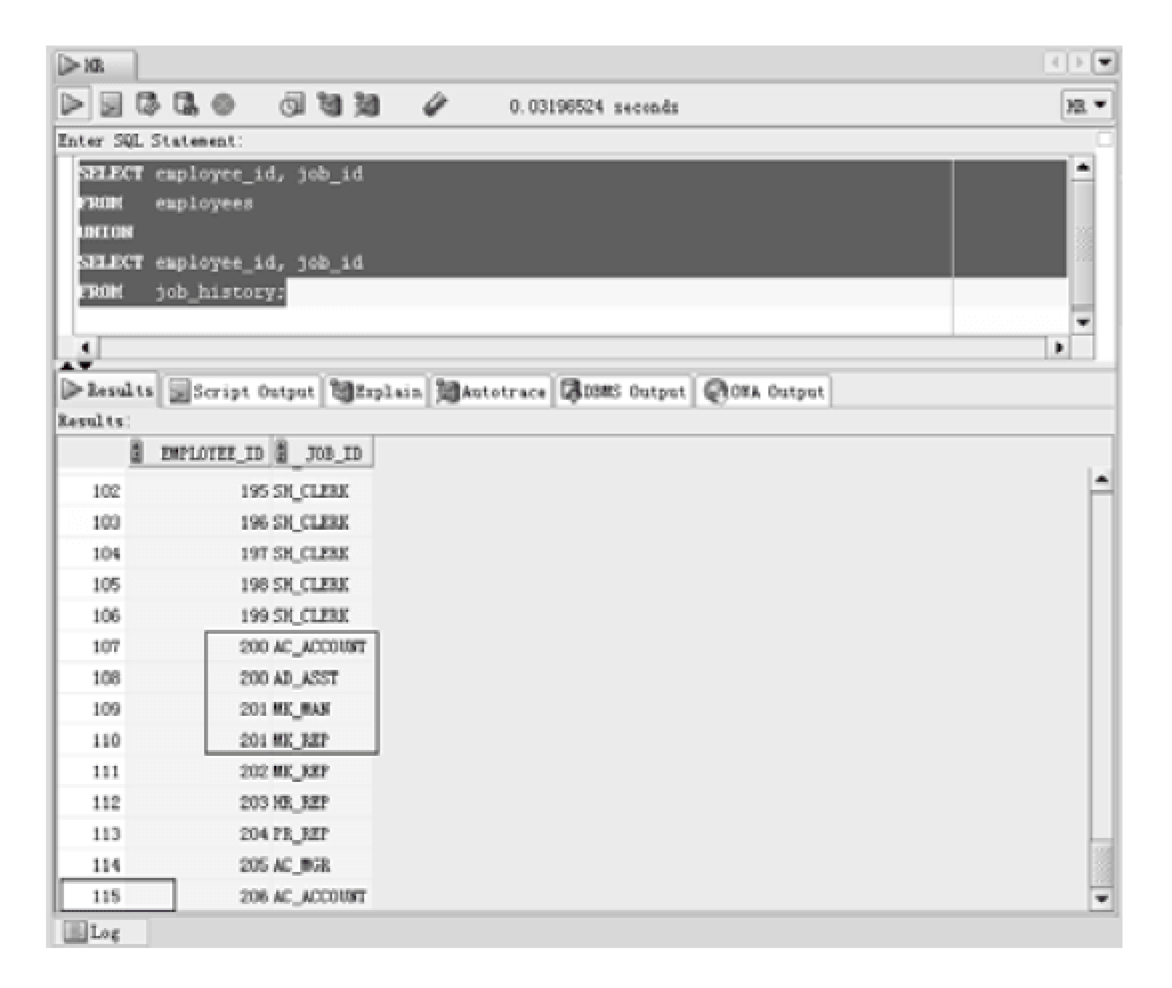


图　17.67

17.6　UNION ALL 集合操作（运算）符

知道 UNION 运算是如何去掉两个查询结果中的重复数据行的吗？是使用排序。设想一下，如果在这一操作中使用的表有几百万行乃至几千万行数据，那么由于排序的原因，UNION 操作将相当慢，而且会严重地影响数据库系统的效率。为了提高 UNION 操作的效率，Oracle 引入了另一个类似的运算符 UNION ALL，其操作示意图如图 17.68 所示。

UNION ALL 操作符将返回两个查询结果的所有数据行并且包含重复数据行。使用这一操作符时所得到的结果可能会有冗余，即可能产生数据的不一致，但是由于它不进行排序操作，所以效率较高。效率和数据的一致性是一个矛盾，作为系统的设计者或开发者，您必须平衡这两者之间的矛盾。永远没有一个完美的设计，在我们所生活的世界中就不可能找到完美的东西。

Oracle 对 UNION 操作所定下的规则也同样适用于 UNION ALL 操作，但是也有两个例外，UNION ALL 操作不去掉重复数据行，并且输出结果也不进行排序。

如果老板只是想要获取所有员工当前担任的和以前所担任过的职位的详细信息，那就可以使用带有 UNION ALL 运算符的查询语句，其操作不但很简单而且效率很高，如例 17-2。

例 17-2

```
SELECT employee_id, job_id
FROM   employees
UNION ALL
SELECT employee_id, job_id
FROM   job_history
ORDER BY  employee_id
```

在 SQL Developer 的 SQL Worksheet 区域中输入例 17-2 的 SQL 语句，然后单击 SQL Worksheet 区域左上角的“执行语句”图标（或按 F9 键），如图 17.69 所示。

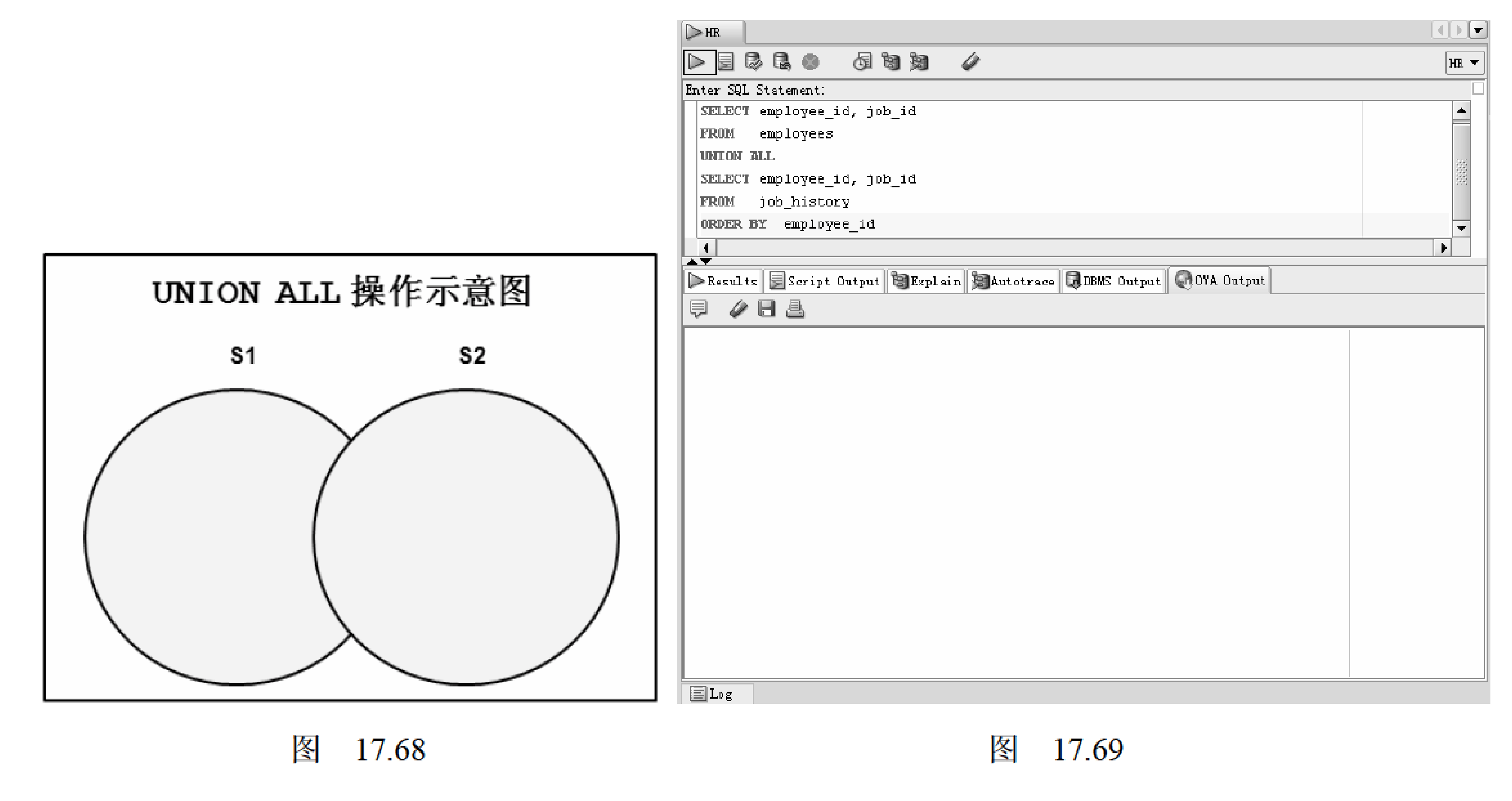


图 17.68

图 17.69

然后，向下滚动滚动条直到看到第 83 和第 84 行数据，如图 17.70 所示。

第 83 和第 84 行的数据相同，它们表明 EMPLOYEE_ID 为 176 的员工在不同的时期内担任过完全相同的职位 SA_REP（销售代表）。而在 UNION 操作中只保留了这两行数据中的一行，因为 UNION 操作要去掉重复行。然后，继续向下滚动滚动条直到最后，如图 17.71 所示。

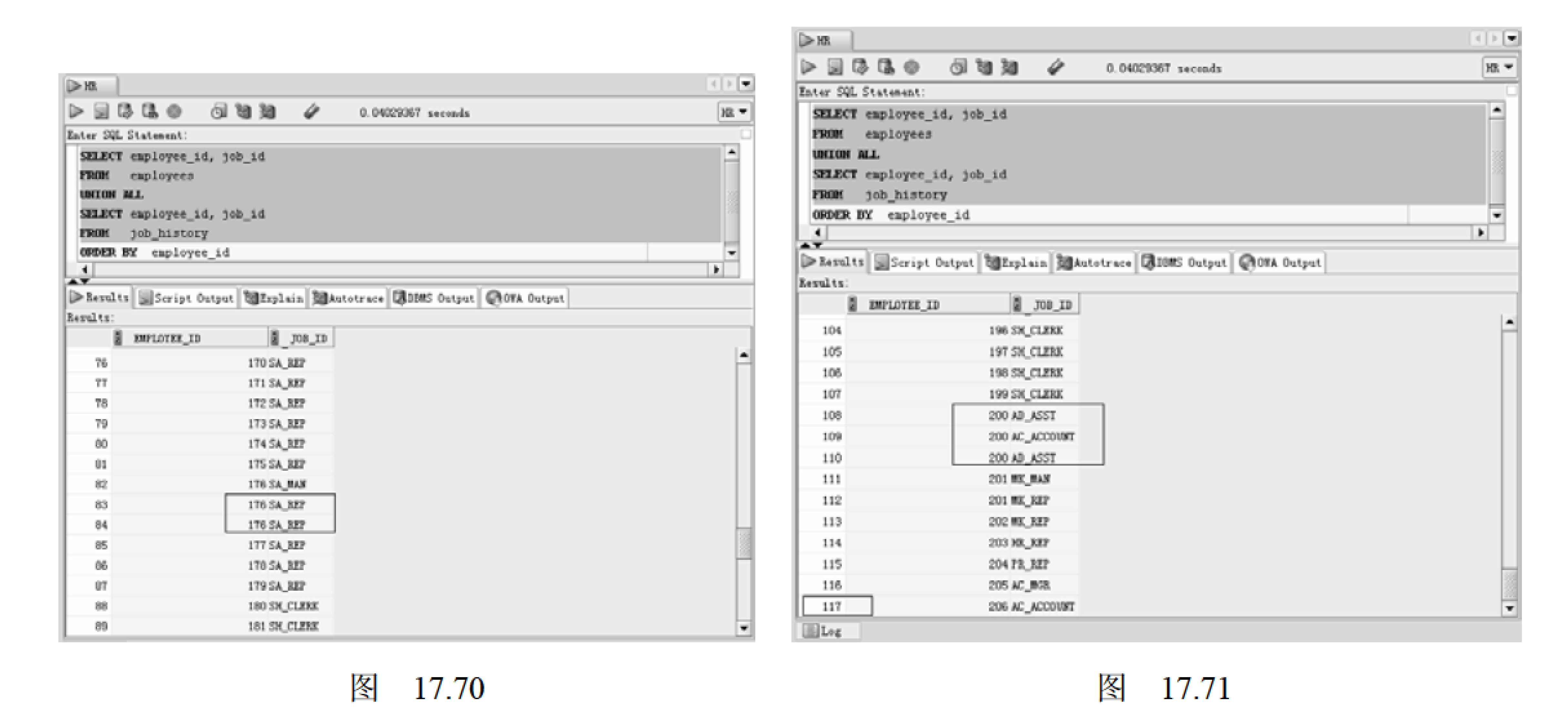


图 17.70

图 17.71

第 108 和第 110 行的数据完全相同，它们表明 EMPLOYEE_ID 为 200 的员工在不同的时期内担任过完全相同的职位 AD_ASST。而在 UNION 操作中只保留了这两行数据中的一行，因为 UNION 操作要去掉重复行。从图 17.71 中还可以得知该 SQL 语句执行的结果共返回 117 行数据，而不是 115 行，因为多了两行重复的数据行。现在读者应该明白 UNION 和 UNION ALL 操作之间的差别了吧？

17.7　INTERSECT和MINUS集合操作（运算）符

INTERSECT 操作符用于返回两个查询结果中所有相同的数据行，图 17.72 是 INTERSECT 操作符操作结果的示意图。

Oracle 也对 INTERSECT 操作做了一些限制，为了使这一操作可以顺利地进行，最好了解如下的规则：

- 在所有查询语句中所选择的列的个数和数据类型必须相同。
- 列名可以不同。
- INTERSECT 并不忽略空值（NULL）。
- 对 INTERSECT（交集）操作之后的表进行反向排序并不改变操作的结果。

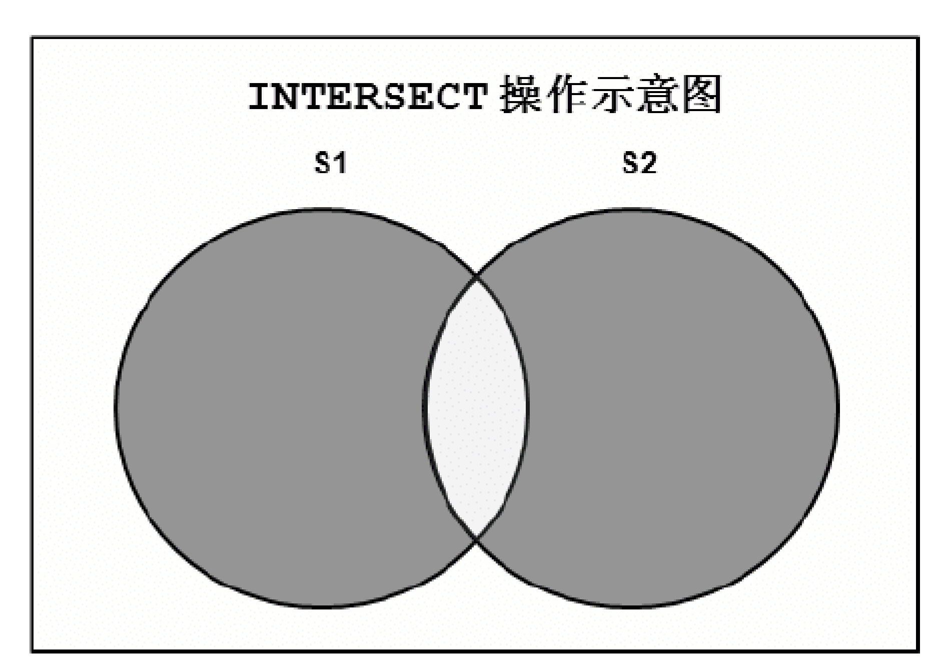


图　17.72

在公司中有这种情况，一个员工可能转换了几个职位，最后又转回到最初的（或之前的）职位。此时，就可以使用 INTERSECT 运算符来完成这样的操作。例如，老板要您打印一份包括员工号（ID）和职位（ID）的报表，而且在这份报表中只包括那些现在所担任的职位上他/她过去也担任过的。您就可以使用带有 INTERSECT 运算符的查询语句方便地完成老板交给的这一光荣使命，如例 17-3。

例 17-3

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history
```

在 SQL Developer 的 SQL Worksheet 区域中输入例 17-3 的 SQL 语句，然后单击 SQL Worksheet 区域左上角的“执行语句”图标（或按 F9 键），如图 17.73 所示。

在 Results 区域会出现两行结果，它们就是 UNION ALL 操作结果中重复的数据行，如图 17.74 所示。

介绍完了 INTERSECT 运算符之后，我们将介绍最后一个集合操作符 MINUS 运算符。MINUS 操作符将返回在第 1 个查询结果中但是不在第 2 个查询结果中的所有数据行（第 1 个查询语句的结果减去第 2 个查询语句的结果），图 17.75 是 MINUS 操作符操作结果的示意图。

Oracle 也对 MINUS 操作做了一些限制，为了使这一操作可以顺利地进行，最好了解如下的规则：

- 在所有查询语句中所选择的对应列的个数必须相同。
- 在所有查询语句中所选择的对应列的数据类型必须属于相同的数据类型组（匹配）。
- 对应列的名字可以不同。

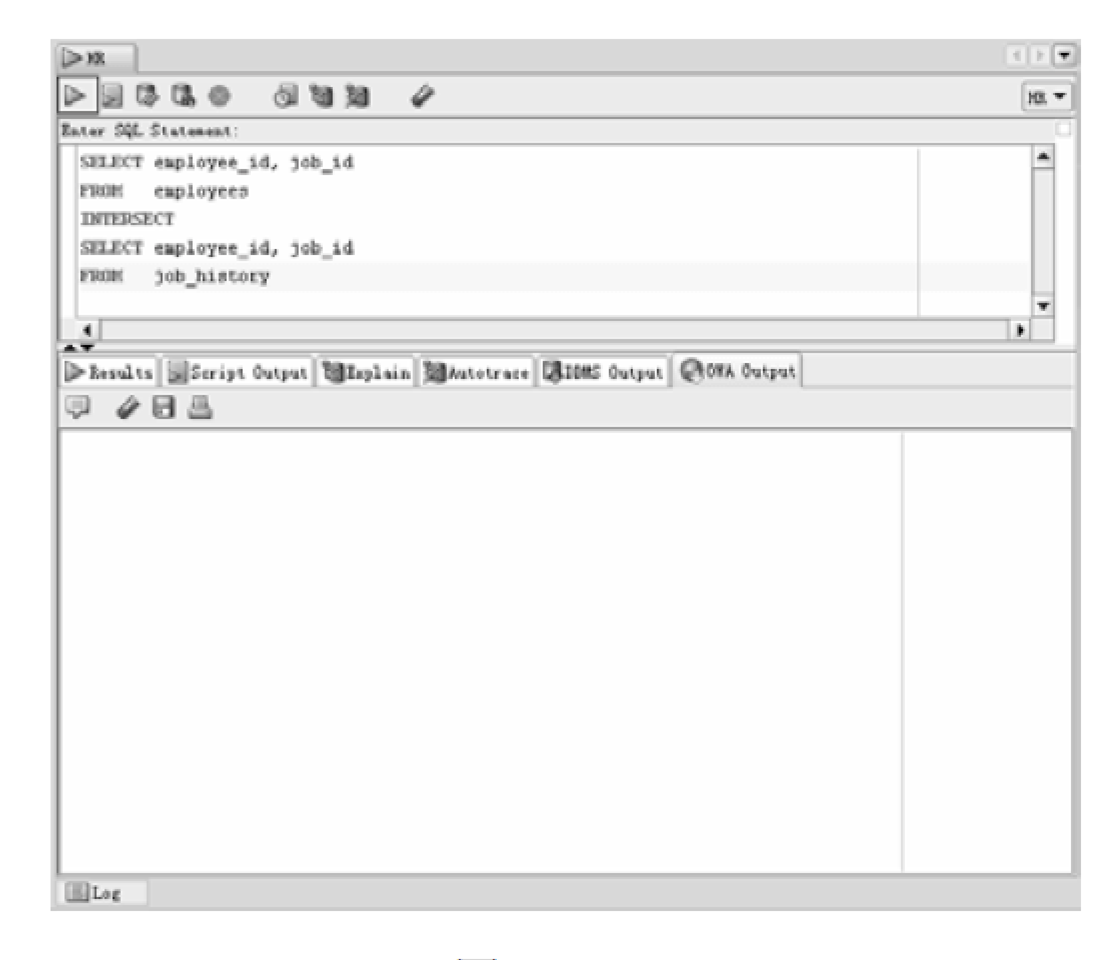


图 17.73

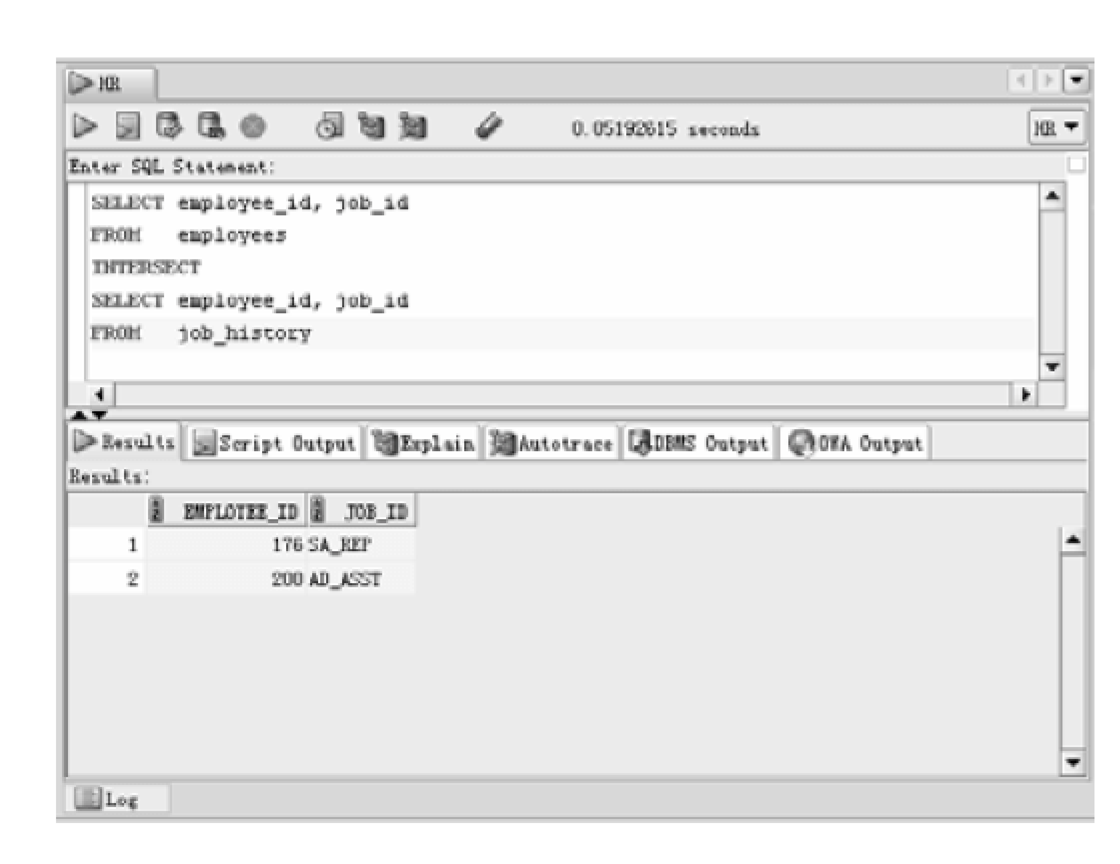


图 17.74

如果老板想知道哪些员工自从进入公司以来职位一直没有变化，此时您就可以使用带有 MINUS 运算符的查询语句方便地满足老板的要求，如例 17-4 所示。

例 17-4

```
SELECT employee_id,job_id
FROM   employees
MINUS
SELECT employee_id,job_id
FROM   job_history
```

在 SQL Developer 的 SQL Worksheet 区域中输入例 17-4 的 SQL 语句，然后单击 SQL Worksheet 区域左上角的“执行语句”图标（或按 F9 键），如图 17.76 所示。

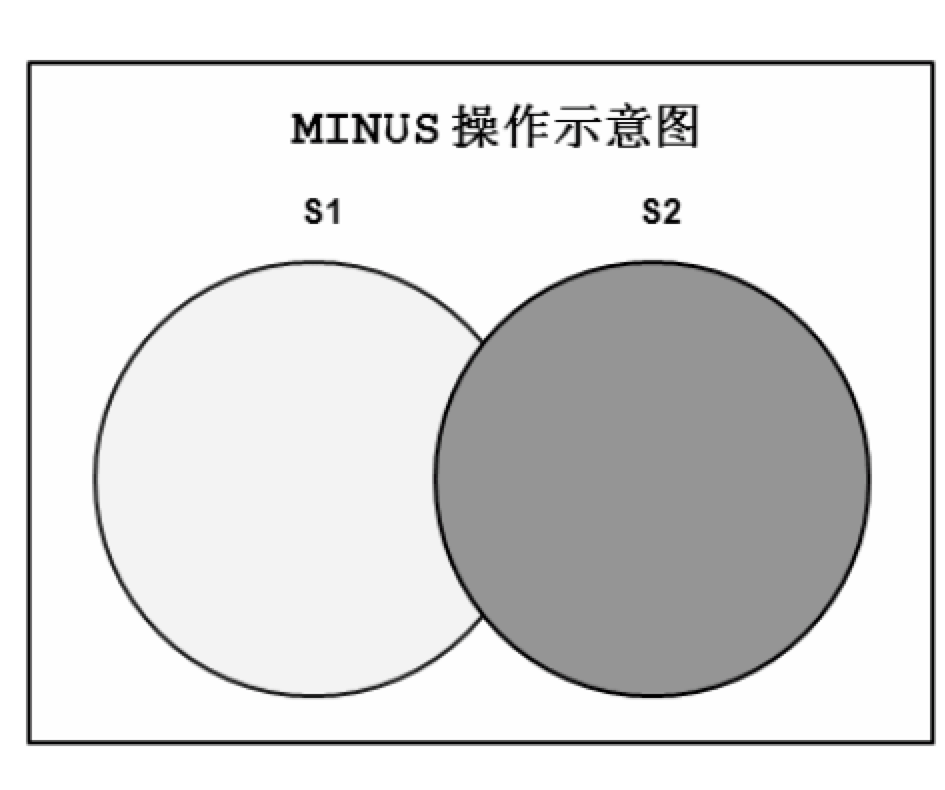


图 17.75

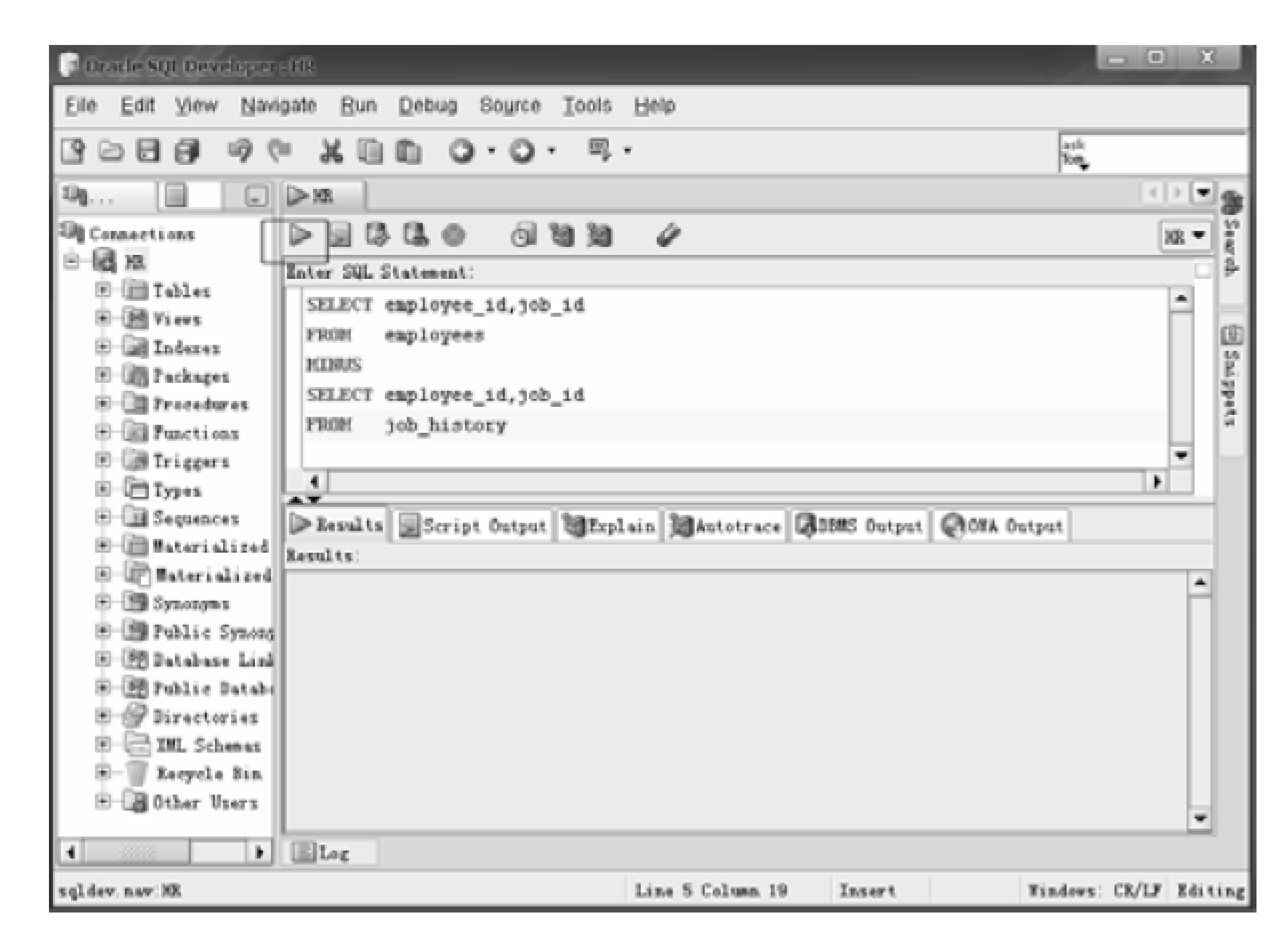


图 17.76

然后，向下滚动滚动条直到最后，如图 17.77 所示。从图 17.77 可以看出，减去了 job_history 表中与 employees 表中共有的数据行之后，查询结果只剩下 105 行数据，即在公司中共有 105 名员工从来就没有转换过工作，他们的员工号（ID）和职位（ID）都显示在 Results 区域。

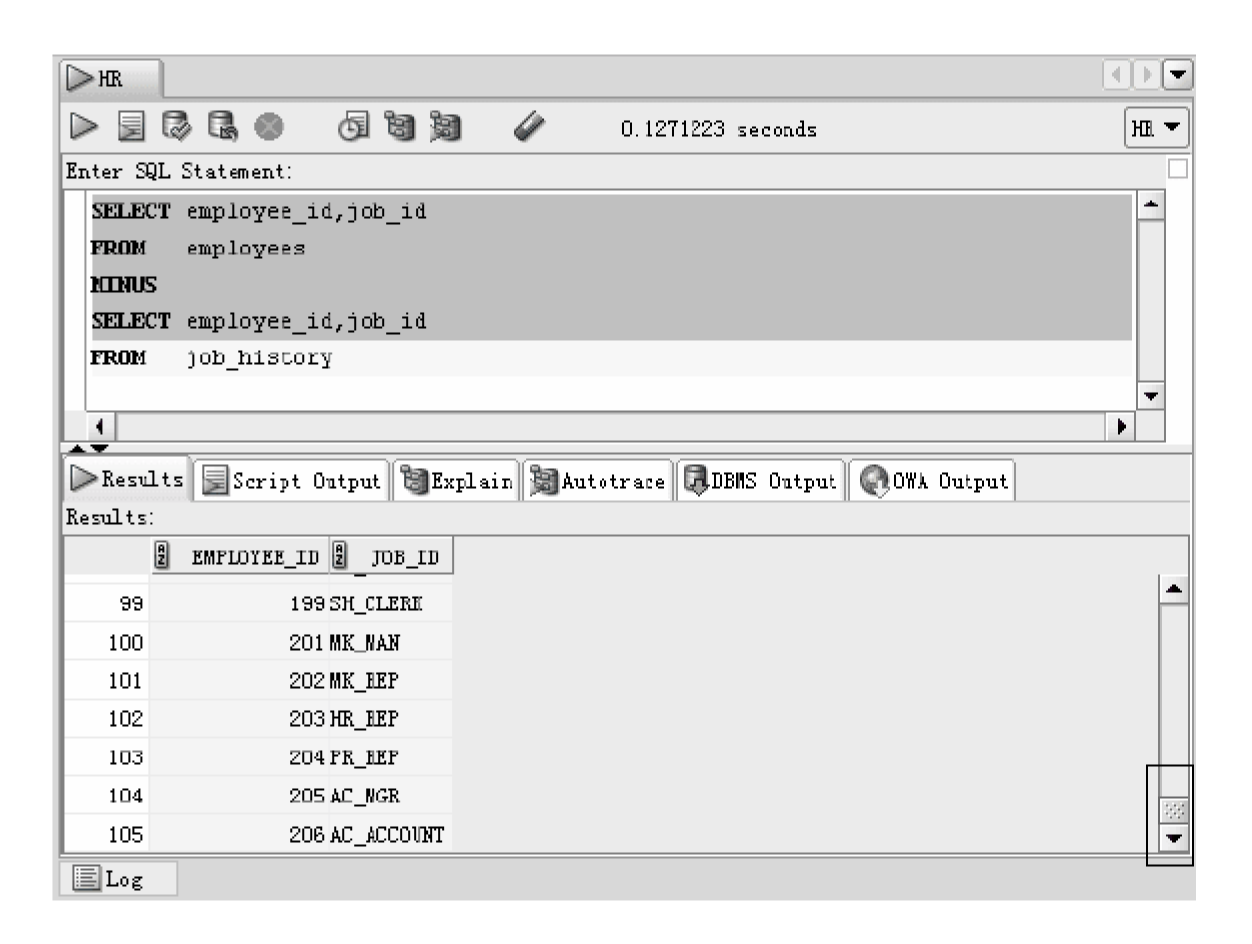


图　17.77

17.8　集合操作（运算）符的特点

在介绍完了 Oracle 的 4 个集合操作（运算）符之后，我们对它们的特点做一个总结。在使用任何集合操作符的复合查询语句中应该遵守如下的原则：

- 每个查询列表中的列（或表达式）的个数和数据类型必须相匹配。
- 集合操作符可以在子查询中使用。
- 如果在 WHERE 子句中使用了带有集合运算符的查询，它们的列数和数据类型就必须与 SELECT 列表中的相同。
- 可以使用括号来改变集合运算符执行的顺序。
- ORDER BY 子句只可以出现在语句的最后面。
- ORDER BY 子句可以接受列名、别名或位置记数法（位置号）。
- 在 ORDER BY 子句中，如果使用了列名或别名，它们必须是第 1 个 SELECT 列表的列名或别名。因此，有时使用位置号可能更方便。

当在查询语句中使用集合操作符时，Oracle 服务器会进行如下的处理：

- 除了 UNION ALL 操作符之外，其他的操作符都将自动删除所有重复的数据行。
- 第 1 个查询语句中列的名字将出现在输出结果中。
- 除了 UNION ALL 操作符之外，输出结果默认按升序排序。
- 如果在两个查询语句中 SELECT 列表的值的数据类型都是 CHAR（定长字符型），其结果的返回值的数据类型也是 CHAR。
- 如果在两个查询语句中有一个 SELECT 列表的值的数据类型为 VARCHAR2（变长字符型），其结果的返回值的数据类型也是 VARCHAR2。

17.9　查询语句的匹配

因为在带有集合运算符的复合查询中，要求每个查询列表中的列（或表达式）的个数

和数据类型必须相匹配，这就带来了一个问题，如果要操作的两个表的列并不匹配，就无法使用集合操作符。为了解决这一问题，我们可以使用人造列和数据类型转换函数使两个查询列表中的列的个数和数据类型相匹配。

假设老板想要一份有关所有部门的清单，并要求该清单上要显示位置号（ID）、部门名和所在的省份。为了完成使命，您首先启动 SQL Developer，然后选择 HR 用户的表 DEPARTMENTS，如图 17.78 所示。

从图 17.78 的显示可以发现，在 DEPARTMENTS 表中只有 LOCATION_ID 列和 DEPARTMENT_NAME 列，而没有与所在省份相关的列。于是，您选择 HR 用户的 LOCATIONS 表，如图 17.79 所示。

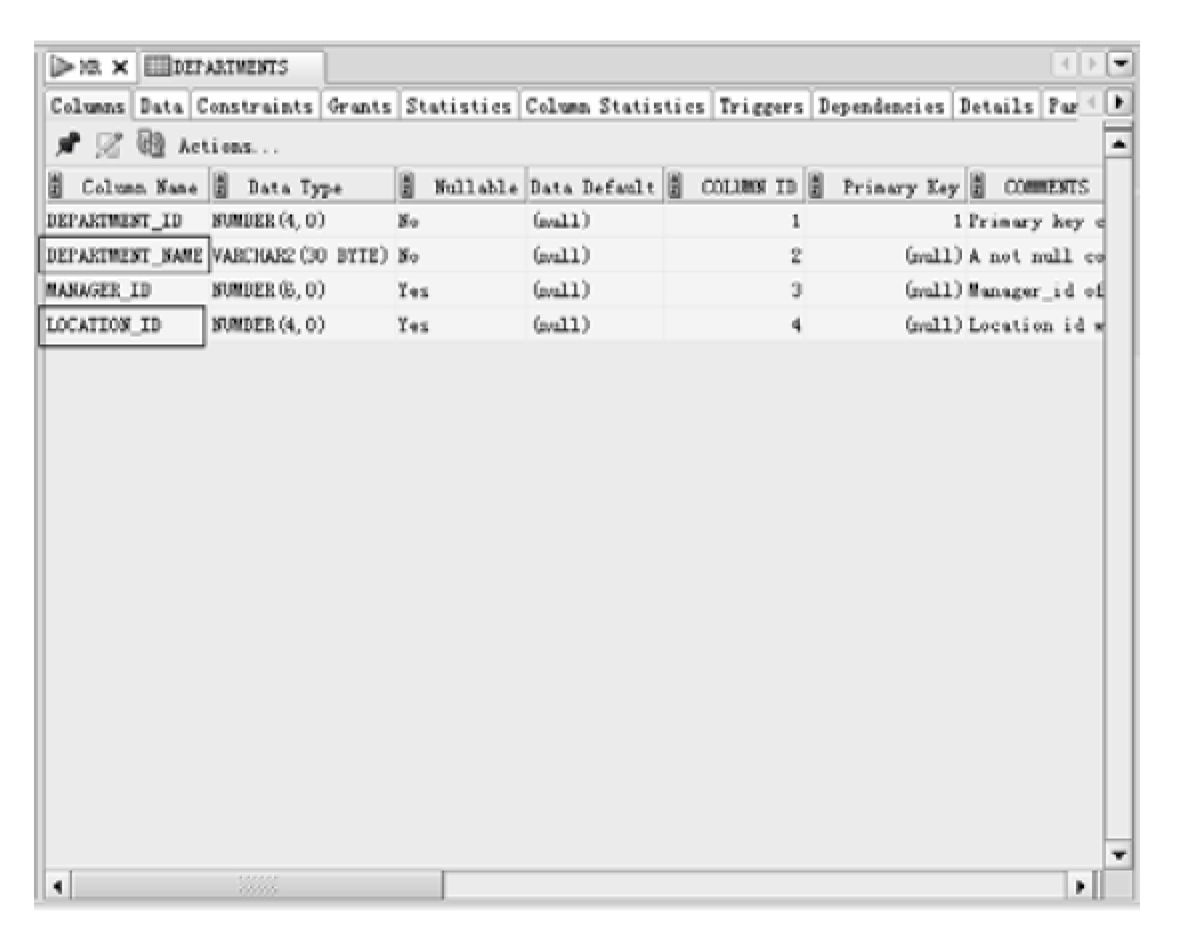


图 17.78

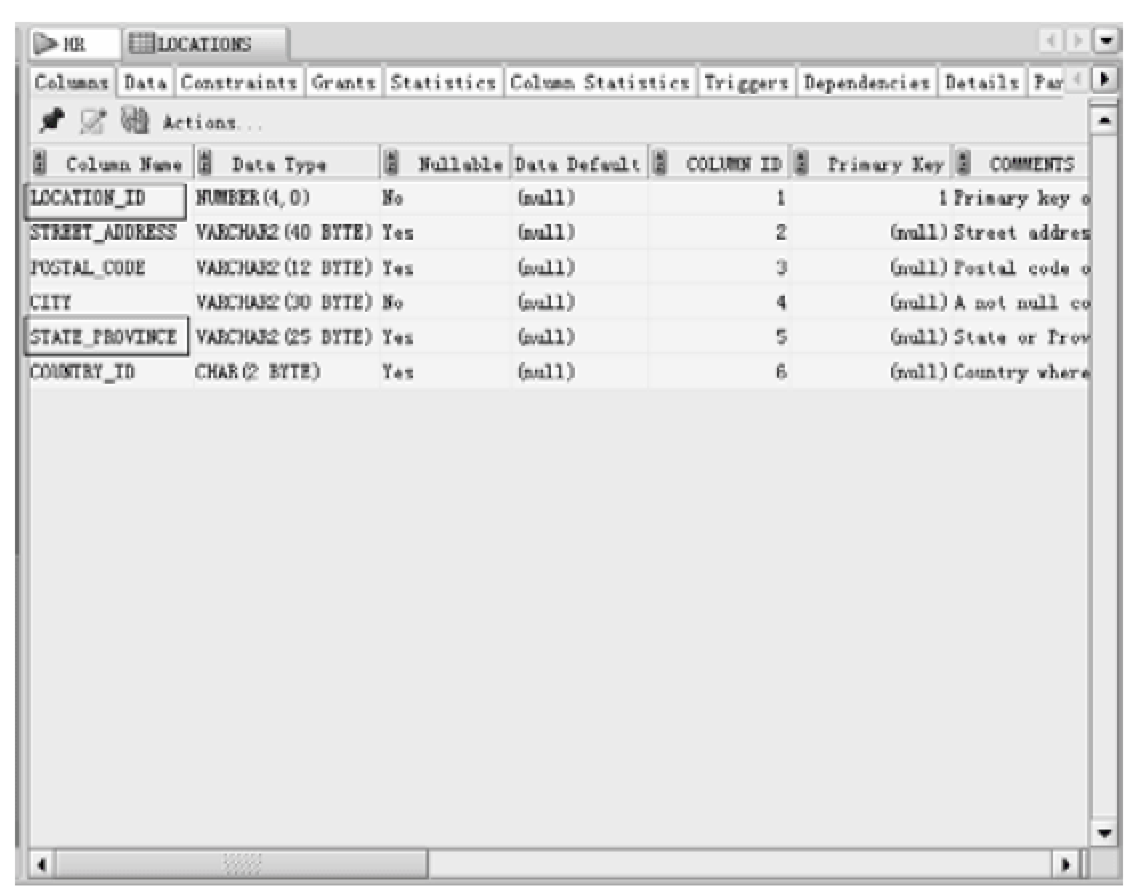


图 17.79

从图 17.79 的显示可以发现，在 LOCATIONS 表中有 LOCATION_ID 列和 STATE_PROVINCE 列，但是又少了 DEPARTMENT_NAME 列。

显然对 DEPARTMENTS 和 LOCATIONS 表进行操作是无法获得老板所需的信息的，于是您想到了刚学会的使用人造列和类型转换函数的方法，写出了如例 17-5 的带有 UNION 操作符的复合查询。因为在 DEPARTMENTS 表中没有 STATE_PROVINCE 列，所以您使用了 TO_CHAR(NULL)（仓库地址为别名）来匹配 LOCATIONS 表中相应的列。同样，因为在 LOCATIONS 表中没有 DEPARTMENT_NAME 列，所以您使用了 TO_CHAR(NULL)（部门为别名）来匹配 DEPARTMENTS 表中相应的列。

例 17-5

```
SELECT location_id, department_name "部门",
   TO_CHAR(NULL) "仓库地址"
FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "部门",
   state_province
FROM locations
```

在 SQL Developer 的 SQL Worksheet 区域中输入例 17-5 的 SQL 语句，然后单击 SQL Worksheet 区域左上角的“执行语句”图标（或按 F9 键），如图 17.80 所示。

然后在 Results 区域将出现该复合查询的结果，如图 17.81 所示。您可以发现在缺少信息的地方都被填入了(null)。

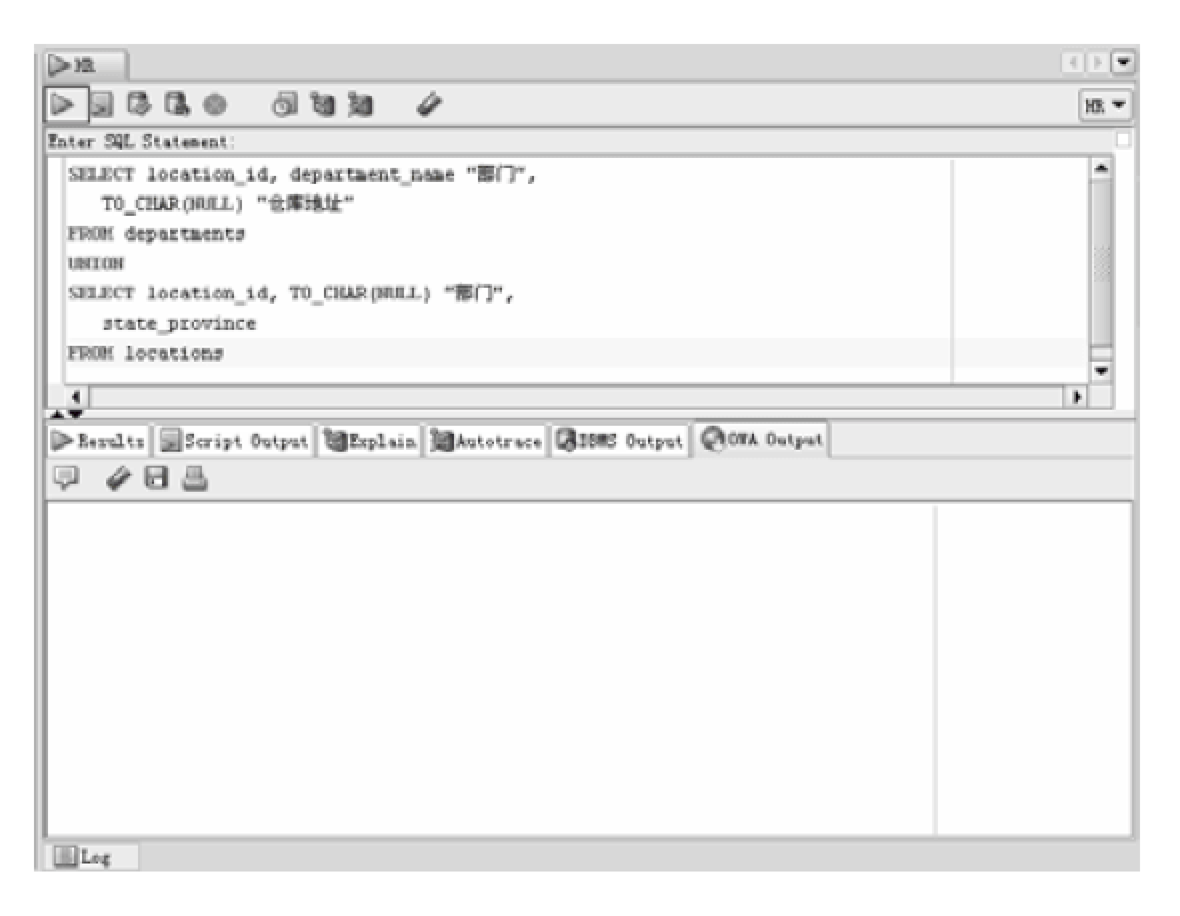


图　17.80

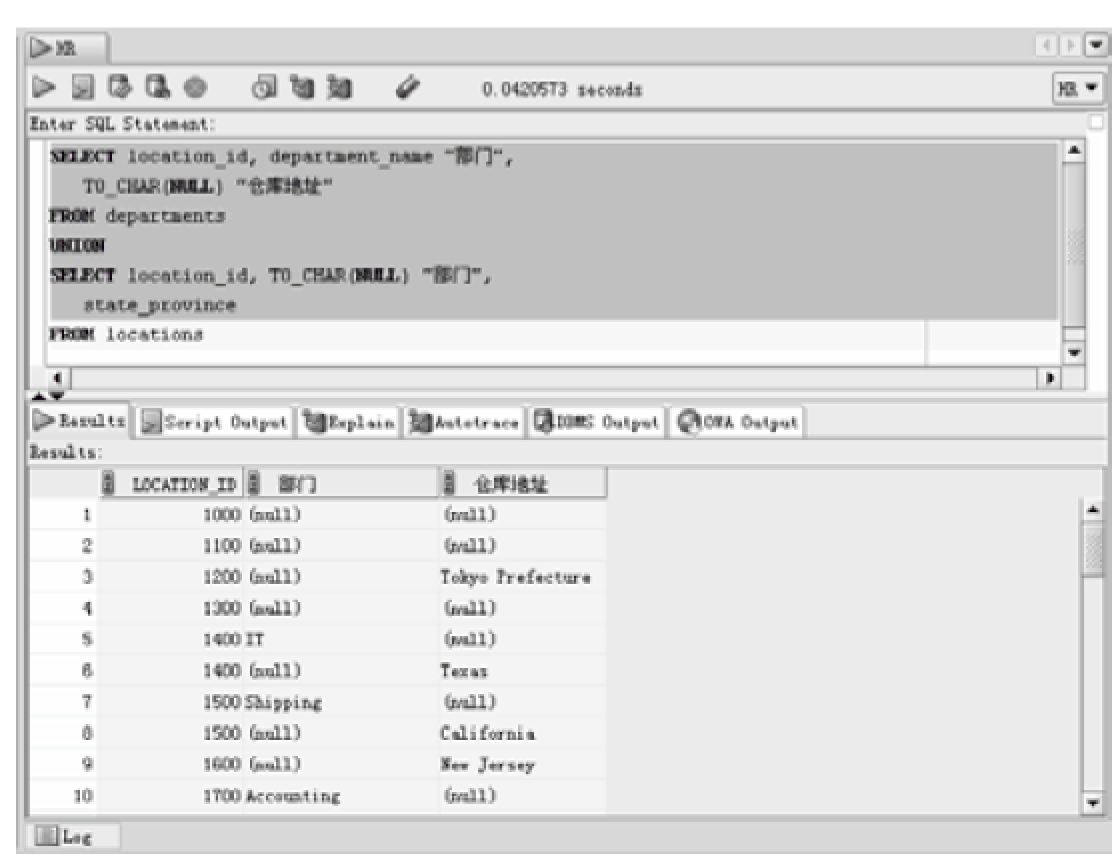


图　17.81

老板又想要一份有关所有员工的清单，并要求该清单上要显示员工号（EMPLOYEE_ID）、姓（LAST_NAME）、所担任的所有职位号（JOB_ID）和工资（SALARY）。为了完成任务，需要对 EMPLOYEES 和 JOB_HISTORY 两个表进行 UNION 操作。但问题是 JOB_HISTORY 表中没有 LAST_NAME 和 SALARY 这两列，于是您又想到了人造列和类型转换，这次您使用了如例 17-6 所示的复合查询语句。

例 17-6

```
SELECT employee_id, last_name, job_id,salary
FROM   employees
UNION
SELECT employee_id, '-',job_id,0
FROM   job_history
```

要注意的是，在例 17-6 的人造列中并未使用数据类型转换函数，而是直接转换成了相应的字符（-）和数据（0），也许这样的复合查询语句看上去更简单、易懂。

在 SQL Developer 的 SQL Worksheet 区域中输入例 17-6 的 SQL 语句，然后单击 SQL Worksheet 区域左上角的“执行语句”图标（或按 F9 键），如图 17.82 所示。

然后在 Results 区域将出现该复合查询的结果，如图 17.83 所示。您可以发现在缺少信息的 LAST_NAME 列都填入了字符“-”，而在缺少信息的 SALARY 列都填入了 0。

通过以上的学习，读者应该已经发现了 Oracle 的 SQL 语言的功能是相当强大的。如前几节的例子，如果不是使用了 Oracle 的集合运算符和相关的功能，而改用其他的高级程序设计语言来完成，您可能需要写很长且很复杂的程序，而使用 Oracle 提供的功能只需一个比较简单的复合查询语句就行了。所以在信息系统的开发中语言的选择是非常重要的，如果选择了合适的语言，您的开发工作将变得简单而轻松。

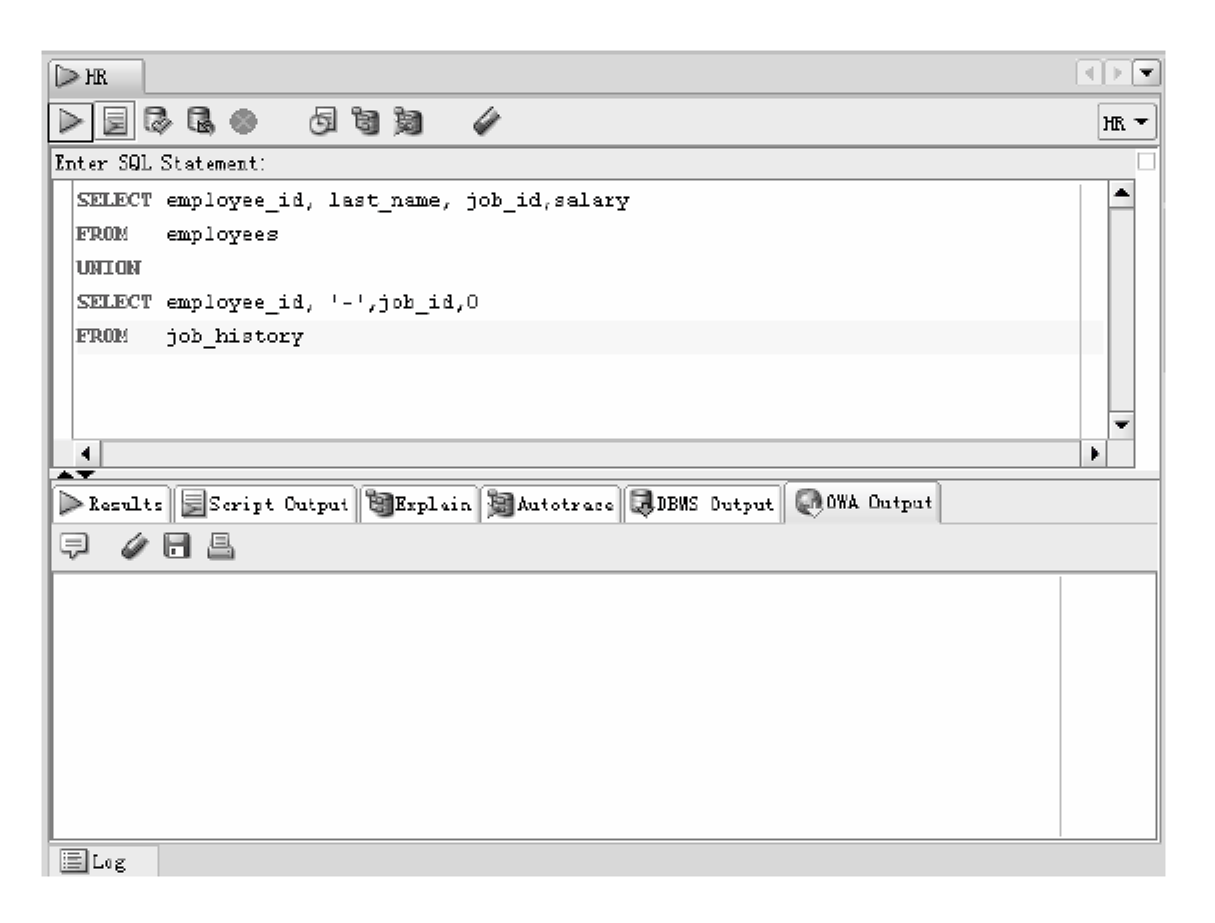


图 17.82

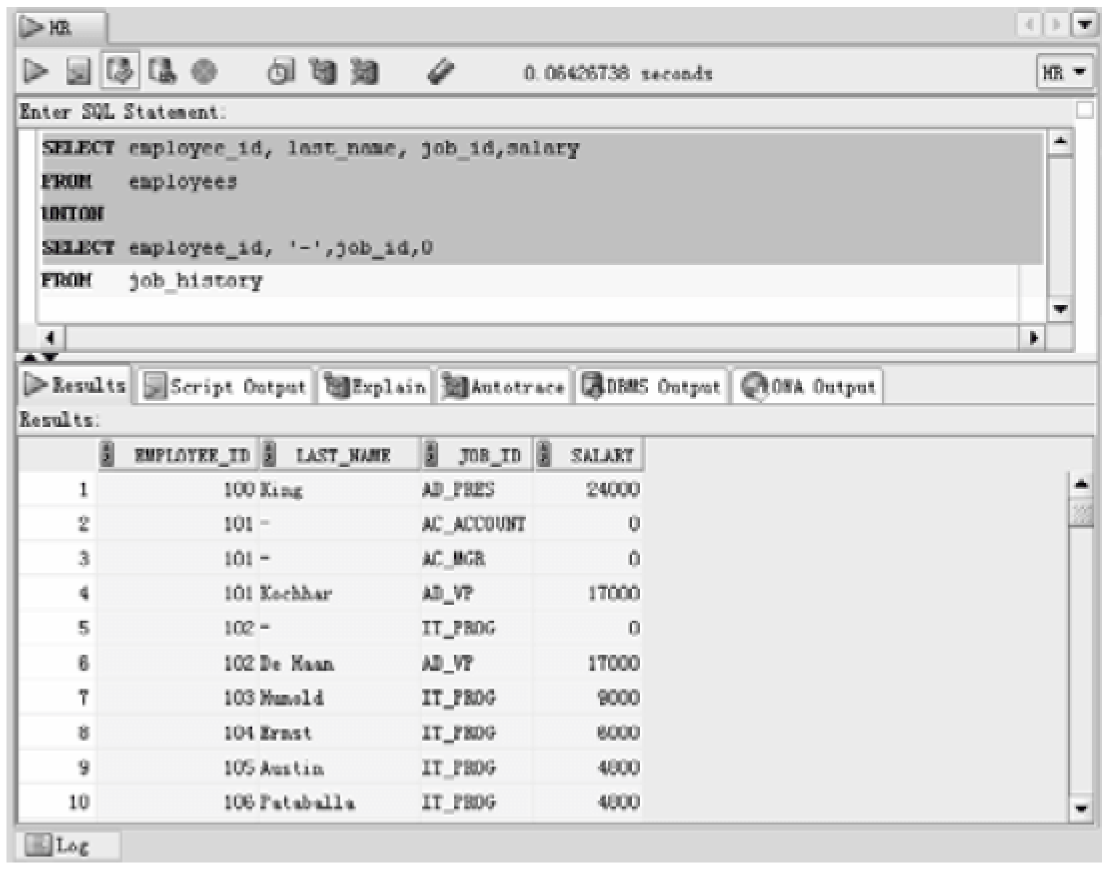


图 17.83

17.10 获取执行计划和控制行的顺序

读者还记得本书第 13.6 节介绍的获取一个 SQL 语句执行计划（步骤）的方法吗？虽然使用所介绍的方法获得了所需的执行计划，但是操作起来并不容易。现在有了 SQL Developer 这一功能强大的开发工具，获取一个 SQL 语句执行计划将变得非常容易。我们通过以下的操作来演示具体的操作步骤：

（1）当 SQL 语句执行成功之后，选择 Explain 选项卡，如图 17.84 所示。

（2）单击“运行执行计划”按钮，如图 17.85 所示，在窗口的下方就会出现该查询语句的执行计划。

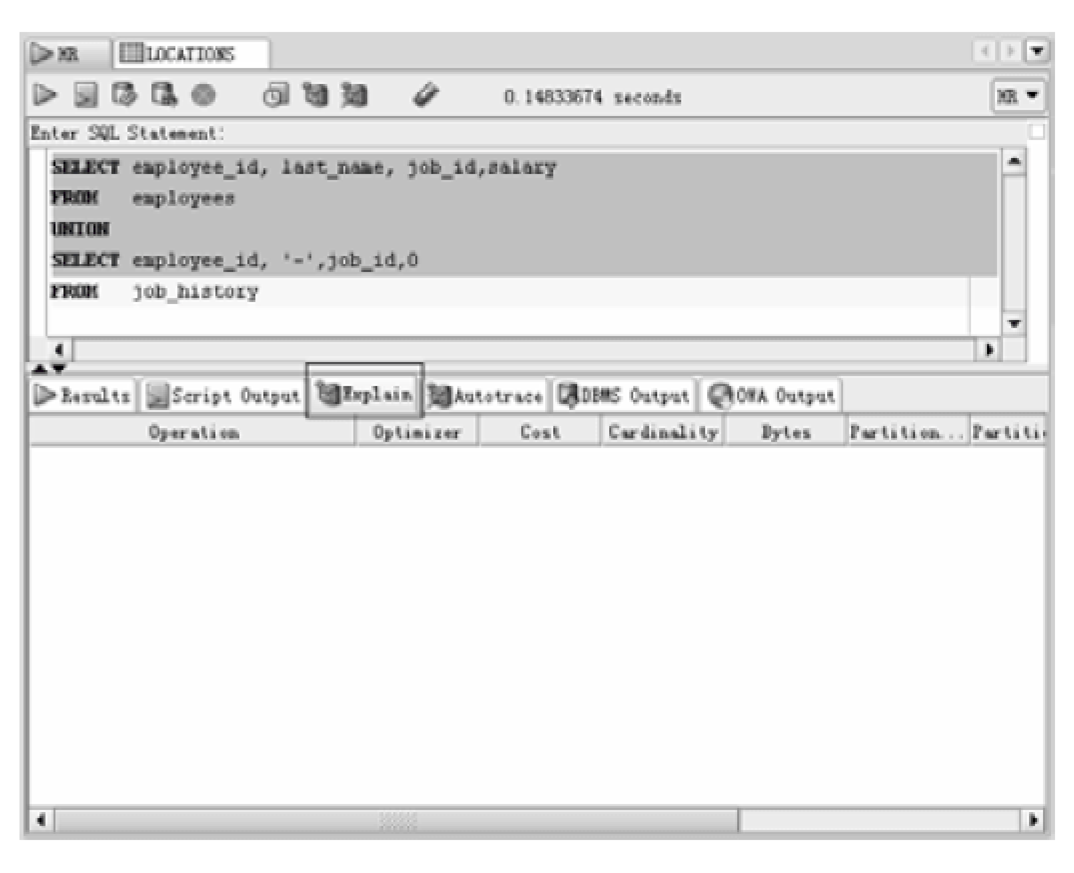


图 17.84

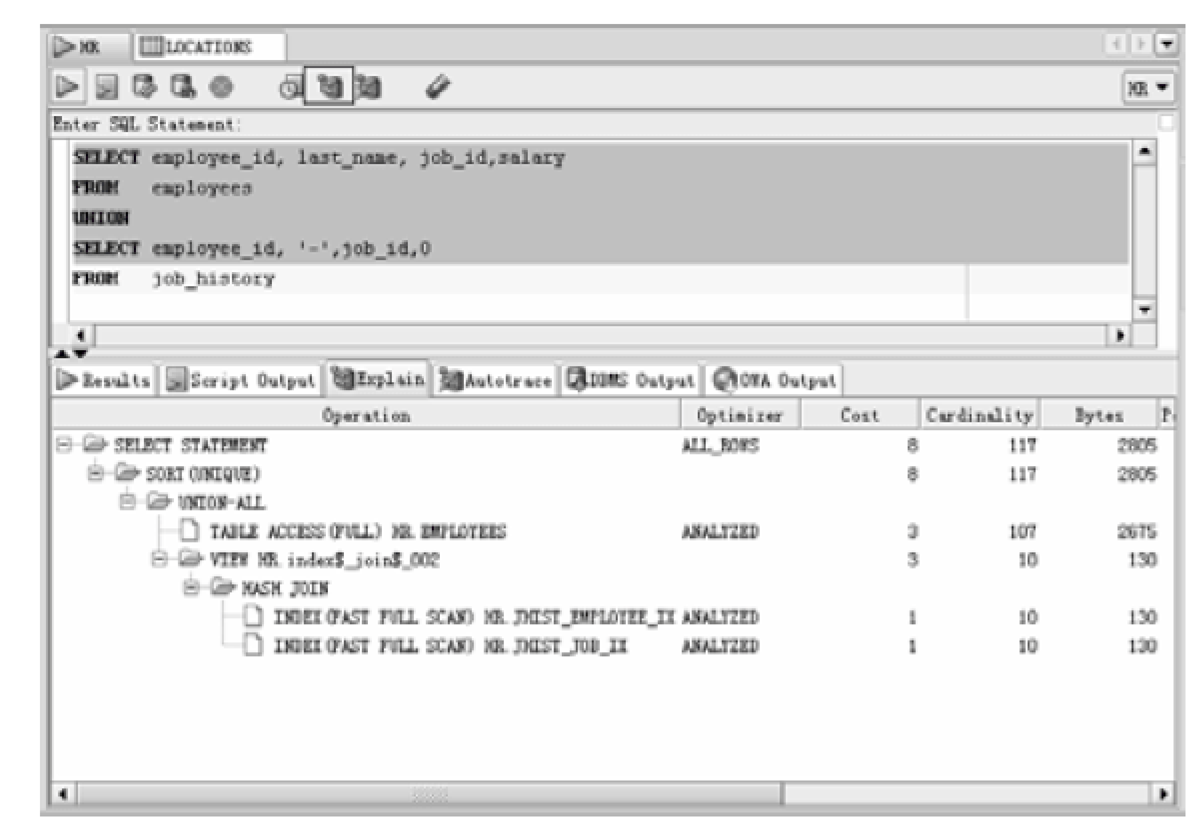


图 17.85

真是“手巧不如家伙妙”，猴子能那么快地进化成人靠的就是工具。有了 SQL Developer 这一功能强大的开发工具，您是不是也很快地从“菜鸟”进化成了“大虾”，实现了跨越式的发展。

在复合查询中是通过 ORDER BY 子句来控制数据行显示的顺序的。在默认情况下，输出的结果是按第 1 列升序排序的。在复合查询中只可以使用一次 ORDER BY 子句。如果使

用了 ORDER BY 子句，该子句一定放在查询语句的最后面。下面我们通过显示一个极具争议的历史、也是文学人物的一些最新考古发现来演示如何控制复合查询结果的数据行显示顺序。

（1）首先将例 17-7 所示的 SQL*Plus 命令和复合查询语句写入记事本并存盘。

例 17-7

```
COLUMN a_dummy NOPRINT
Define v1 = 您知道历史上真正的苏妲己吗？她是一位受到诋毁最多的女性，是一位完全被妖魔化的女性。
Define v2 = 她用自己的美貌、个人魅力和机智勇敢彻底颠覆了汤商王朝，是建立大周朝的第一功臣。
Define v3 = 事实上，她是事业上最成功的女性，也是一位最敬业的女性，是有史以来最出色的女间谍。
Define v4 = 极为讽刺的是，她对大周朝的赤胆忠心和卓越功绩换来的却是被顶天上司处死和千古骂名。
Define v5 = 因为周武王和姜太公不想因为她的出色表现而损害了他们精心营造出的大周朝清明的声誉，
Define v6 = 那些开国元勋们也不愿让她在大周朝胜利的大饼中再分去一大块，结果是自己人都盼她死。
Define v7 = 其实，姜子牙掩面斩妲己，不是因为她美，而是作为她的顶头上司，他无颜面对自己的下属。
Define v8 = 最后，周朝的最高决策层只能编造出来狐狸精附体这样的弥天大谎来哄骗天下和她的家人。
Define v9 = 妲己一案向人们展示了周朝辉煌历史中一个最阴暗的角落，也展示了政治和人性肮脏的一面。
Define v10 = 妲己在爱情上可以说是一个彻底的失败者，为了事业，先后失去了两个真爱的人伯邑考和帝辛。
Define v11 = 要敬畏历史。因为透过历史，我们可以看到政治中最肮脏的角落和人性最丑陋的内心深处！！！

SELECT '&v1' AS "历史人物真相大揭秘", 1 a_dummy
FROM dual
UNION
SELECT '&v2', 3
FROM dual
UNION
SELECT '&v3', 2
FROM dual
UNION
SELECT '&v4', 4
FROM dual
UNION
```

```
SELECT '&v5', 5
FROM dual
UNION
SELECT '&v6', 6
FROM dual
UNION
SELECT '&v7', 7
FROM dual
UNION
SELECT '&v8', 8
FROM dual
UNION
SELECT '&v9', 9
FROM dual
UNION
SELECT '&v10', 11
FROM dual
UNION
SELECT '&v11', 10
FROM dual
ORDER BY 2;
```

☞ **指点迷津：**

在例 17-7 中，COLUMN a_dummy NOPRINT 表示 a_dummy 列不打印（显示），利用 a_dummy 可以随意地控制复合查询结果的数据行显示顺序。在这个例子中之所以使用替代变量而不是直接将字符串放入 SQL 语句中，是为了修改和调试的方便。如果将来要显示潘金莲的新发现，就只需要修改替代变量的定义部分，而 SQL 语句不需做任何修改。

（2）启动 SQL*Plus，并以 hr 用户（也可以是其他用户）登录 Oracle 数据库，如图 17.86 所示。

（3）在 SQL*Plus 中输入例 17-8、例 17-9 和例 17-10 的命令对显示进行格式化，并关闭显示替代变量值。

例 17-8

```
set line 120
```

例 17-9

```
set pagesiz 40
```

例 17-10

```
set verify off
```

（4）然后将例 17-7 中的所有命令复制到 SQL*Plus 中并运行，就会得到如下的显示输出。

```
历史人物真相大揭秘
----------------------------------------------------------------------
您知道历史上真正的苏妲己吗？她是一位受到诋毁最多的女性，是一位完全被妖魔化的女性。
事实上，她是事业上最成功的女性，也是一位最敬业的女性，是有史以来最出色的女间谍。
她用自己的美貌、个人魅力和机智勇敢彻底颠覆了汤商王朝，是建立大周朝的第一功臣。
极为讽刺的是，她对大周朝的赤胆忠心和卓越功勋换来的却是被顶天上司处死和千古骂名。
因为周武王和姜太公不想因为她的出色表现而损害了他们精心营造出的大周朝清明的声誉，
那些开国元勋们也不愿让她在大周朝胜利的大饼中再分去一大块，结果是自己人都盼她死。
其实，姜子牙掩面斩妲己，不是因为她美，而是作为她的顶头上司，他无颜面对自己的下属。
最后，周朝的最高决策层只能编造出来狐狸精附体这样的弥天大谎来哄骗天下和她的家人。
妲己一案向人们展示了周朝辉煌历史中一个最阴暗的角落，也展示了政治和人性肮脏的一面。
要敬畏历史。因为透过历史，我们可以看到政治中最肮脏的角落和人性最丑陋的内心深处!!!
妲己在爱情上可以说是一个彻底的失败者，为了事业，先后失去了两个真爱的人伯邑考和帝辛。

已选择 11 行。
```

从以上的显示输出可以看出，a_dummy 列确实没有显示，而且数据行也按我们的要求重新排列了。集合操作符的功能强大吧？接下来，我们使用 SQL Developer 运行例 17-7 的命令，看看与使用 SQL*Plus 之间有什么不同。

（5）启动 SQL Developer，将例 17-7 中的命令复制到 Enter SQL Statement 区域后单击“执行语句”图标（或按 F9 键），如图 17.87 所示。



图　17.86

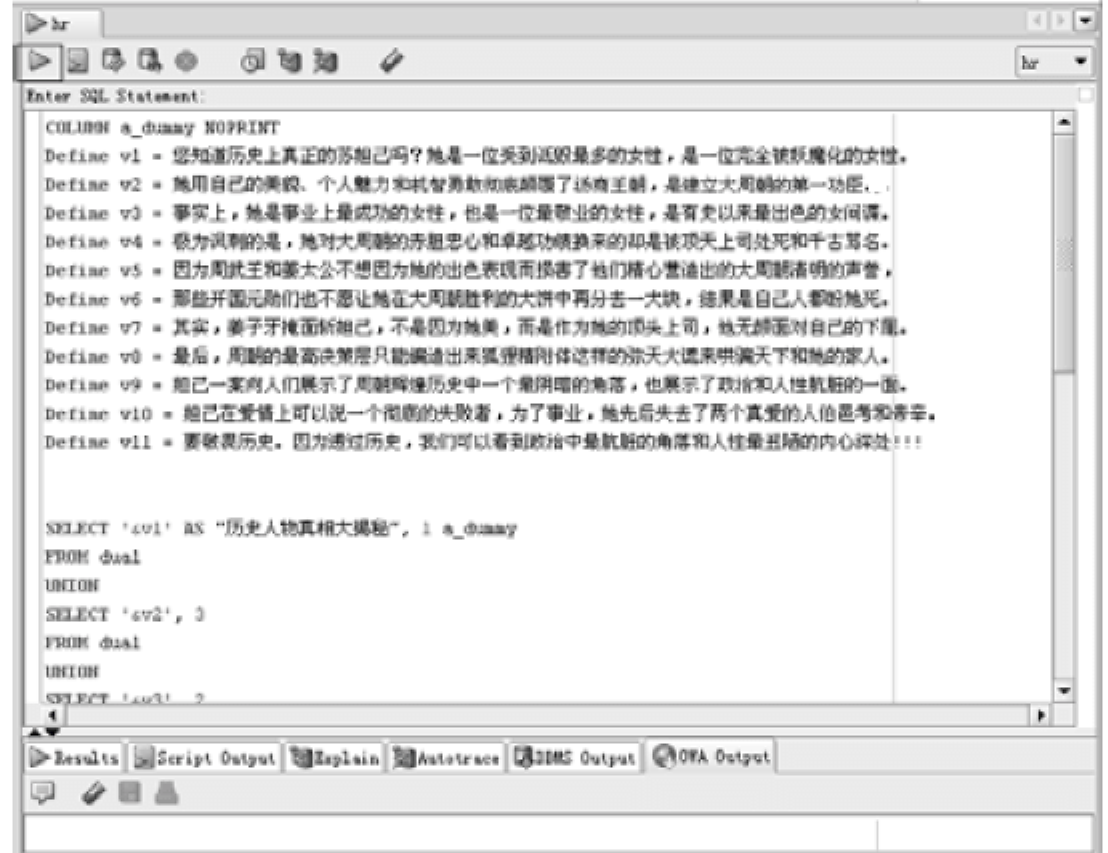


图　17.87

（6）然后会出现要求输入 SQL*Plus 替代变量的窗口，如图 17.88 所示。为了简单，所有弹出的对话框都单击“取消”按钮关闭。

（7）单击“运行脚本”图标，如图 17.89 所示。这次将会得到复合查询语句执行的正确结果，但是 a_dummy 列还是照常显示。

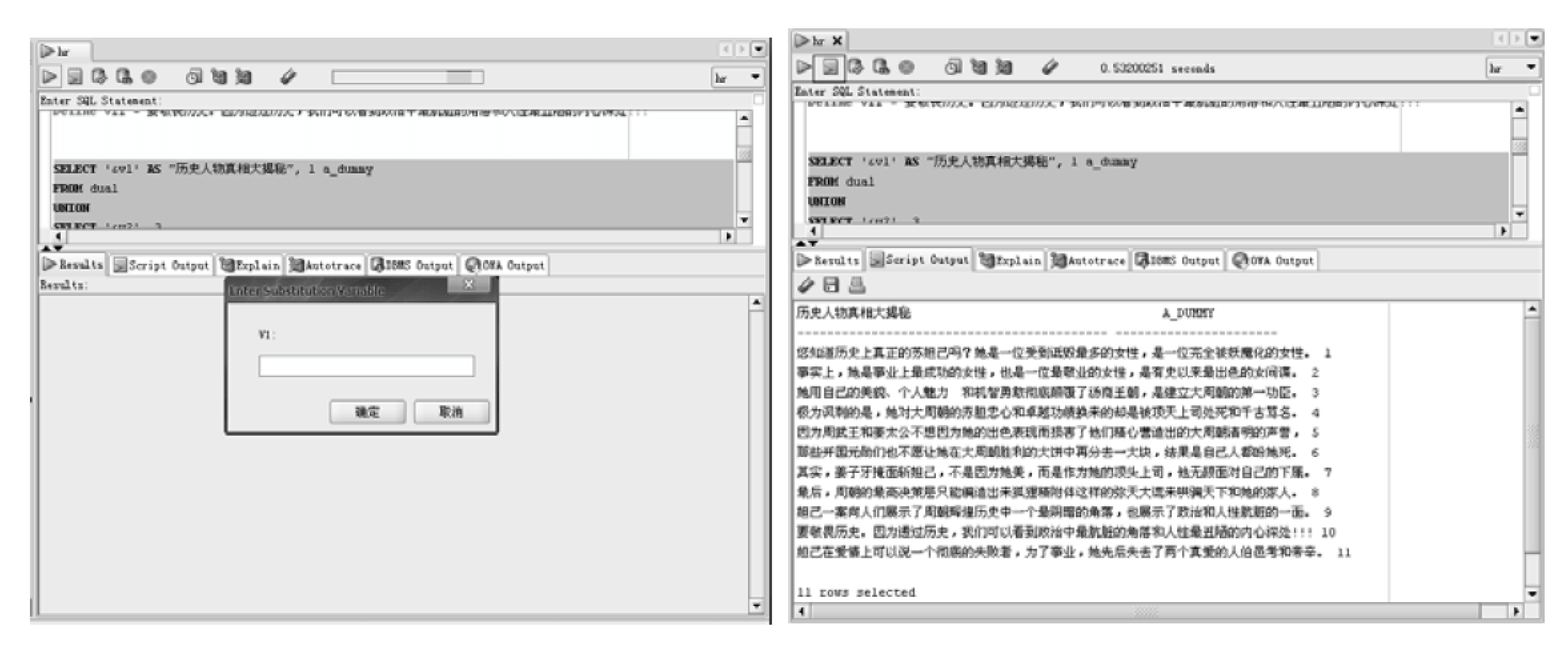


图　17.88　　　　　　　　　　　　　　图　17.89

（8）如果退回到之前的界面，再次单击“执行语句”图标（或按 F9 键），将会直接得到正确的结果，但是 a_dummy 列也照常显示，如图 17.90 所示。这是因为所有的 SQL*Plus 替代变量都已经定义了。

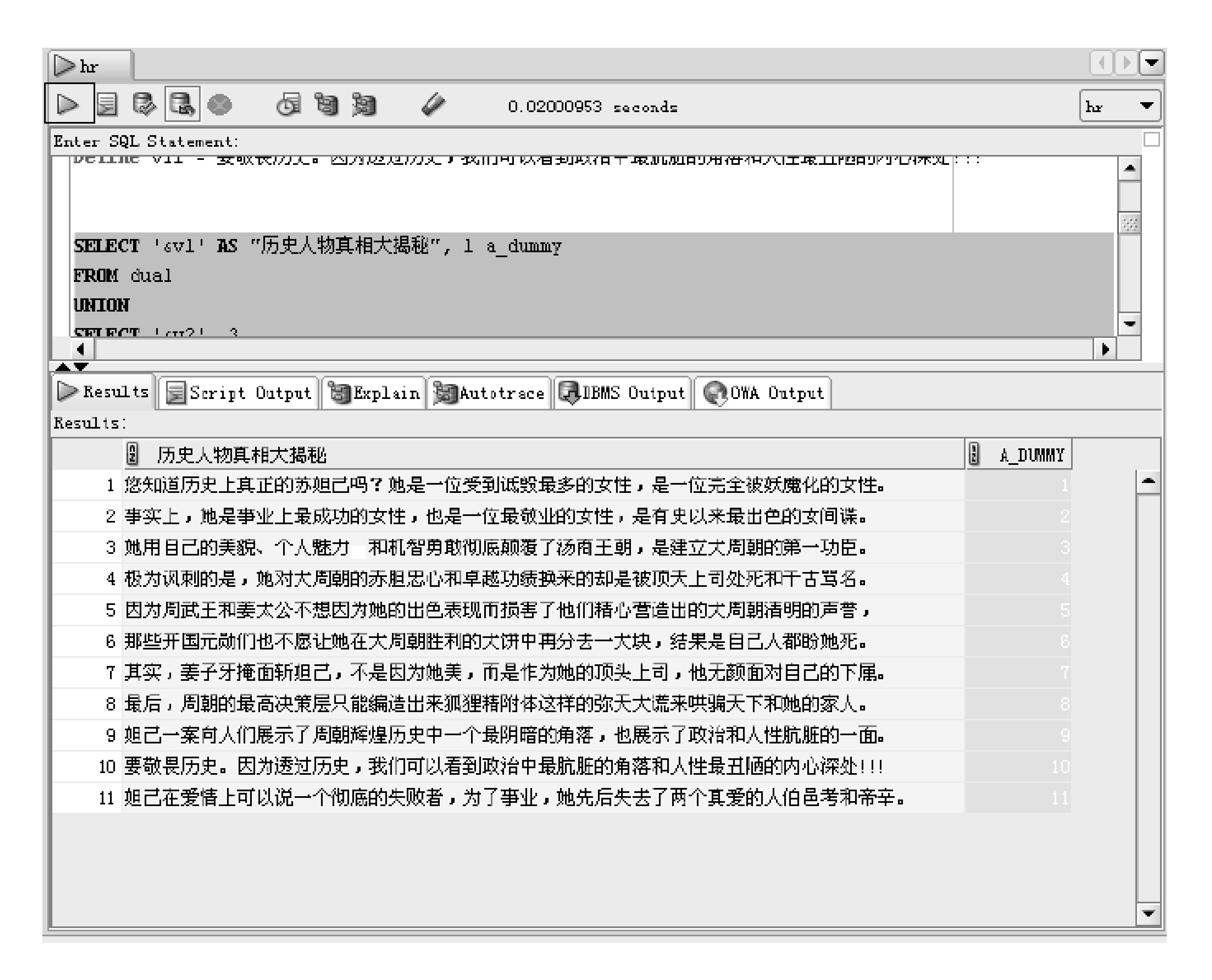


图　17.90

通过以上的例子，读者应该知道在 SQL Developer 中使用 SQL*Plus 命令并不能产生预期的结果。另外，SQL Developer 是一个开发工具，它的功能过于强大，因此，在生产环境中最好限制它的使用，否则可能会产生意想不到的灾难性的后果，就像导弹发射的按钮不能谁都能按一样。

读者在了解了这位有史以来事业上最成功的女强人苏妲己的悲惨结局之后，不知还有没有勇气为事业的成功而继续废寝忘食地拼搏。历史的前进、文明的进步总是要付出代价的，不但会有人战死在疆场，也要有人蒙冤死去。

第18章

Express 概述和安装

通过前面 SQL 的学习，读者应该已经能够将所需的信息输入 Oracle 数据库并且使用 SQL 和 SQL*Plus 命令对这些信息进行管理和维护了。但是如何使那些没有任何数据库或计算机知识和背景的用户在未经培训的情况下就能方便地使用这些数据呢？显然使用 SQL 和 SQL*Plus 命令对这样的人群是不可能的。可能有的读者已经想到了网页，让用户使用在互联网上冲浪的方法访问和使用这些数据应该是最佳的方案，也是眼下最时髦的。

在 Oracle 10g 之前将 Oracle 数据库中的数据做成网页放在互联网上并不轻松，因为使用 SQL 不能进行直接的网页编程而必须借助于其他的程序设计语言或工具。而且在使用这些语言或工具之前经常不得不完成一些繁琐的系统配置。Oracle Application Express（快速 Web 应用开发工具）的诞生将使这些令人不愉快的噩梦成为历史。使用 Oracle Application Express 只需一些简单的鼠标单击、拖拉或极少的输入就可以将 Oracle 数据库中的数据以优美的网页形式轻松地展示给任何用户，如图 18.1 所示的订单系统、图 18.2 所示的客户追踪系统和图 18.3 所示的简单商业智能系统。

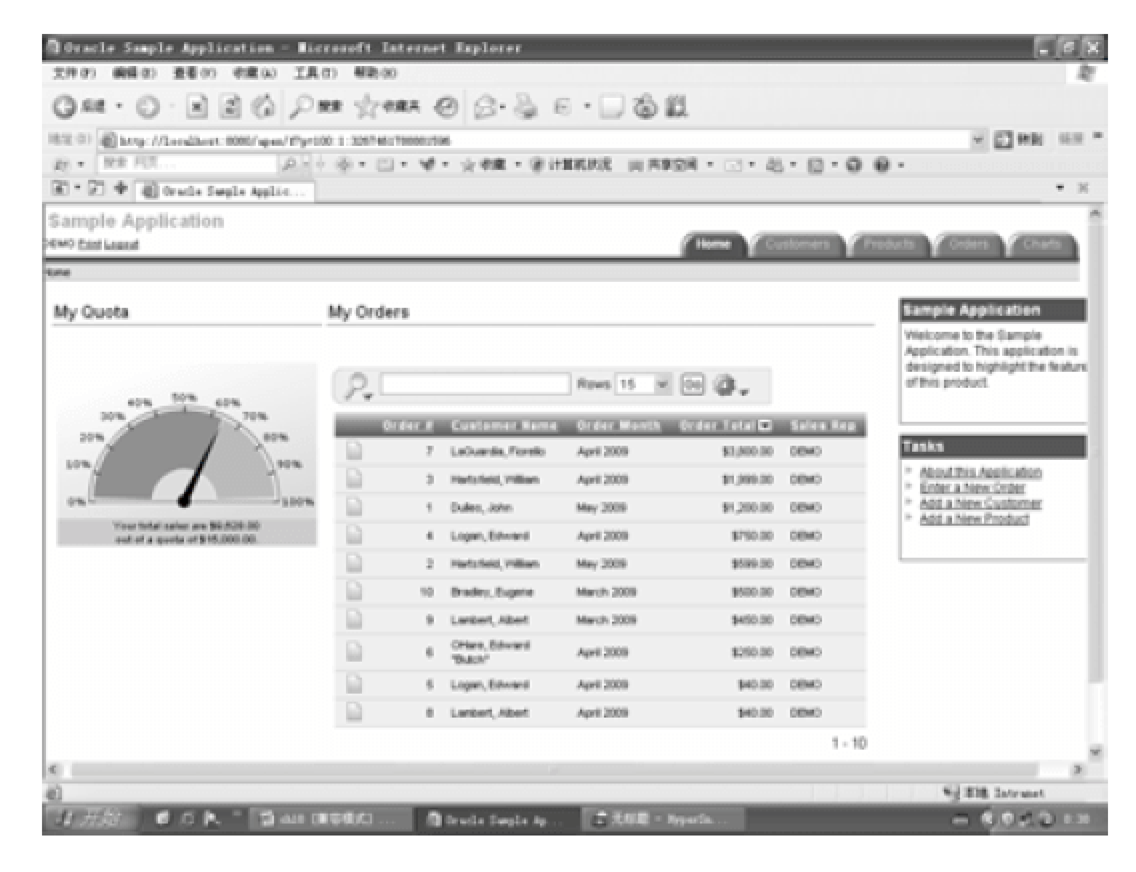


图 18.1

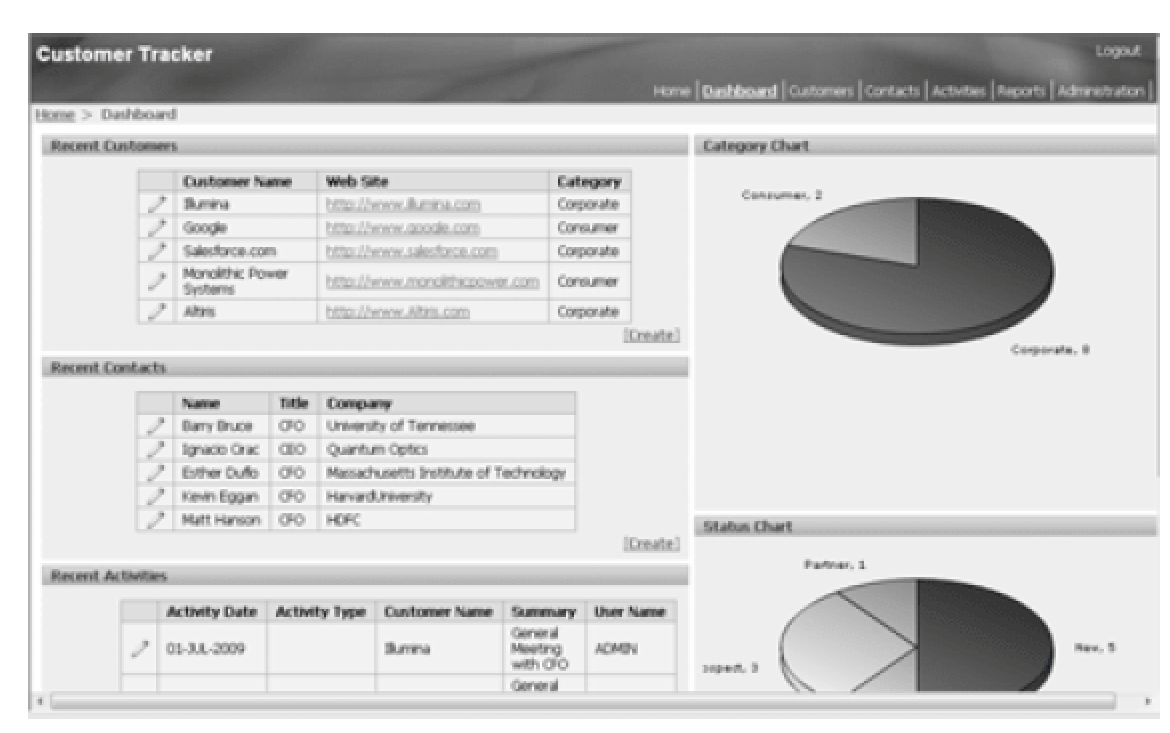


图 18.2

☞ 指点迷津：

如果读者对以下的内容学习起来比较吃力，请不用担心，不懂也没关系，只要按照书中的方法安装就行了，不会影响后面的学习。因为安装和配置 Oracle Application Express 是数据库管理员的职责，以后读者如果学习了 Oracle DBA 课程之后会发现它的安装和配置并不难。其实，多数 Oracle Application Express 的教材并未包括安装和配置的内容，这些教

材都是假定 Express 已经安装和配置好了。但是为了方便读者自学，本书还要详细地介绍 Express 的一种简单和实用的安装和配置全过程。如果读者对这方面的内容特别感兴趣，也可以参考相关版本的 Oracle Application Express Installation Guide，这一文档可以在 Oracle 公司的官方网站免费下载。

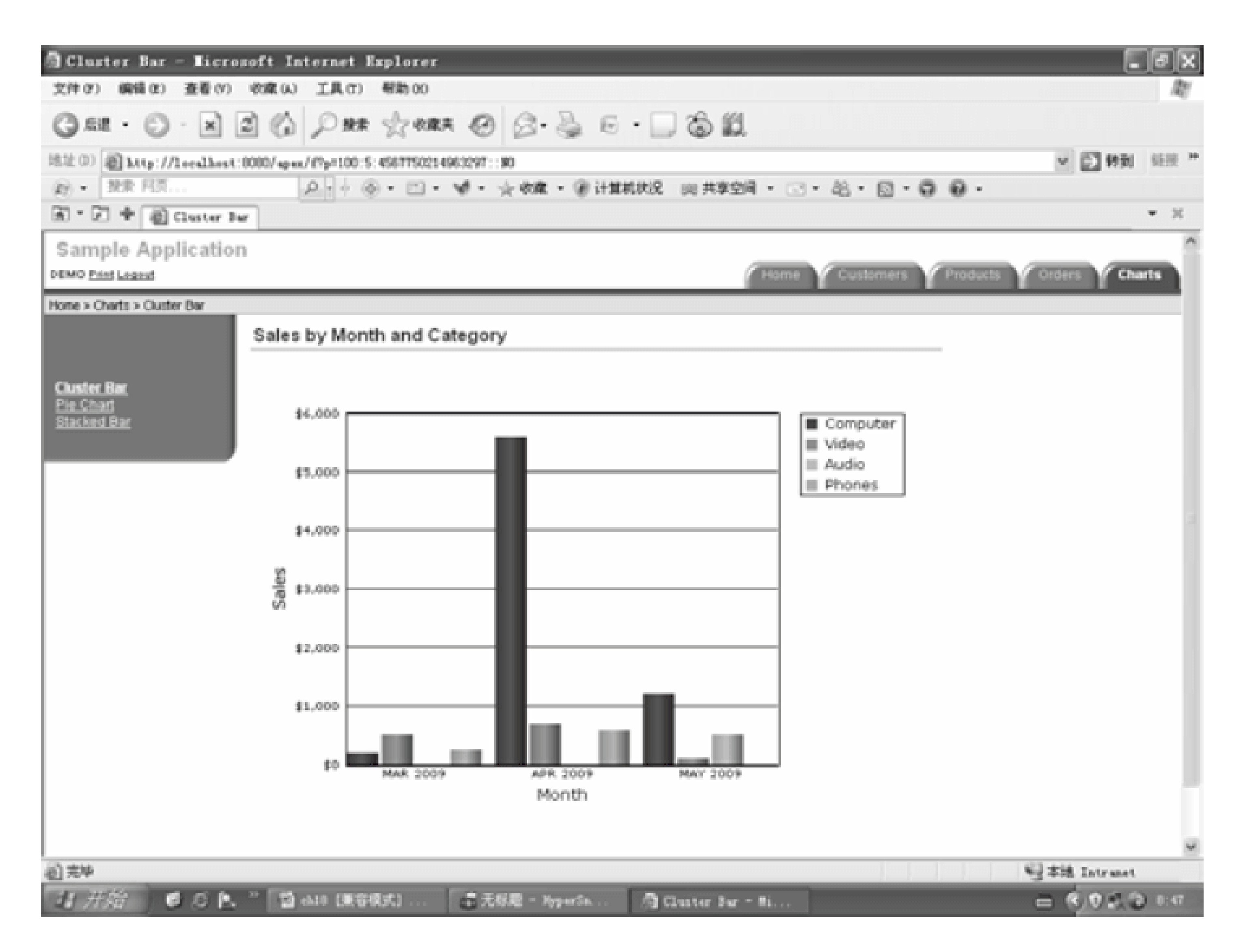


图　18.3

18.1　Oracle Application Express简介

Oracle Application Express（快速 Web 应用开发工具）是为 Oracle 数据库设计的一种基于网络浏览器的快速应用开发工具。它的前身叫 Oracle HTML DB，是 Oracle 10g 引入的。但是在过去的几年当中，这一方便而且功能强大的系统开发和部署工具并未得到迅速的推广，其主要原因是在早期的版本中，Oracle HTML DB 的安装是比较复杂的，并且它消耗的系统资源也很大。随着 Oracle 11g 的推出，Oracle Application Express（以下简称 Express）的安装已经简单了许多。同时，伴随着硬件价格的持续下降和功能的不断提高，它的资源消耗也不是什么问题了。

Express 是一种声明类型的开发工具，它适合于以数据库为中心的环球网应用程序的开发与部署。只需要使用网络浏览器和很少的（甚至没有）编程经验，用户就可以使用这一工具开发出快捷、安全、同时具有专业水平的应用程序。

由于 Express 是被集成在 Oracle 数据库中的，因此，它可以自动地使用数据库的所有特性，如安全、大规模性和可移植性等。当用户使用 Express 时，只需根据系统的提示遵循简单的开发过程就行了，因此，用户可以把他们的注意力集中在解决业务问题而不是编程上。

与其他面向环球网的开发工具不同，Express 不是传统意义上的代码生成器。尽管 Express 为应用程序产生最终的 HTML 代码，但是在创建和修改应用程序期间，它并不产

生任何代码或可执行程序。应用程序的定义被存储在元数据资料库中，在创建应用程序的每一步，Oracle 系统都将创建和修改相关的元数据，并将其存入资料库的表中。当运行一个应用程序时，Oracle 系统会自动完成所有必要的操作和转换，而呈现给用户的就是精美的网页。

18.2　Oracle Application Express的诱人之处

首先，Express 是一个很容易使用的应用程序开发环境，用户只需具有基本的（甚至没有）SQL 知识就可以开发环球网应用程序。与传统的自上而下的过程化或面向对象的程序设计方法不同，Express 是一种基于声明框架结构的互联网应用程序开发环境。用户只需定义页面、用于用户界面控制所引用的共享模板和页面上的接口组件这样的组合，Oracle Application Express 引擎就存在于 Oracle 数据库之中，这个运行期间的引擎将有效地处理应用程序中的 SQL 请求。

SQL 和 PL/SQL 程序员掌握 Express 的学习曲线很短，这无疑减少了企业的培训和招聘成本。利用声明方式来组装应用程序不但使开发的速度加快，而且仅需要很少的开发人员就能完成。Oracle Application Express 能够使企事业充分地利用他们在 SQL 和 PL/SQL 技能方面的现有投资。

Express 具有简单和自身包含的体系结构。因为 Express 引擎是直接嵌在 Oracle 11g 数据库中的，所以用户可以将 Oracle Application Express 安装在一台 PC 上。另外，Express 的开发环境可以支持多个开发人员甚至多个企事业同时进行开发。这使得 Express 的管理员可以集中管理和维护开发环境，并且可以在一个安装上创建一个共享工作组的数据库服务。

利用主题和模板提供了灵活优美及有个性的页面选择。主题提供了一种机制，它可以方便地为整个机构范围内的所有页面提供外观和操作一致的显示。主题既可以为一个应用程序的多个页面共享，也可以由多个不同应用程序的页面所共享。主题是由一些模板组成。使用标准的 HTML 和替代字符串，用户可以通过修改模板来控制页面、报表和其他用户接口元素的外观。

Oracle Application Express 独立于硬件和操作系统。因为 Oracle Application Express 的应用程序的定义是以元数据的方式维护的，并且它们都存在于 Oracle 数据库中，所以整个应用程序可以很容易地从一个 Oracle Application Express 实例（系统）导出，之后再导入另一个实例。Oracle Application Express 可以运行在绝大多数操作系统平台上。

18.3　可以使用Express完成的工作

通过使用 Oracle Application Express（快速 Web 应用开发工具）所提供的特性，用户可以方便地完成如下工作：

- 以交互的方式创建、管理 Oracle 数据库对象和数据。

- 用很短的时间开发出基于互联网应用程序的接口。
- 加强互联网应用程序的安全。
- 使用所提供的大量功能来加强应用程序等。

以下就是这些功能的较为深入的解释。

（1）以交互的方式在后台 Oracle 数据库中创建、查看和管理数据库的对象并操作对象中的数据。Express 是通过一些基于互联网的接口（工具）来完成与后台数据库的交互的，例如，

- 数据加载/卸载（早期叫数据车间）：通过这一工具可以快速地将电子表格或正文文件中的数据输入到数据库的表中，也可以快速地将数据库中的数据导出。
- SQL 命令处理器：通过这一工具可以执行 SQL 查询或执行数据定义语言（DDL）以创建、更改、丢弃数据库的对象。

（2）在向导的帮助下，用很短的时间开发出基于互联网应用程序的接口。其接口（生成器）包括，

- 报表（生成器）：可以很容易地生成报表来进行格式化 SQL 查询的结果，用户在建立报表时只需了解一些基本的 SQL 知识即可。
- 表单（生成器）：Express 提供了一些预定义的常用表单。用户可以利用它们方便而快速地生成表单，并且可以利用它们查询数据库中的数据或对数据进行 DML 操作，还可以使用表单对数据进行有效性检查和共享已有的值列表等，也可以通过表单向用户提供字段一级的帮助。
- 图表（生成器）：为 SQL 查询的结果生成各种预定义的图表，如直方图或饼图。
- 向导：Express 中包含了多个向导，通过使用这些向导用户可以非常容易地创建应用程序。
- 日历：通过使用日历可以生成基于某个表单的日历。

（3）除了开发应用程序之外，还可以通过使用如下的安全机制来加强应用程序的安全。

- 验证方案：验证方案是一种核实一个用户的身份并通知 Oracle Application Express 引擎这个登录是否成功的方法。应用程序将使用这一登录信息。Oracle Application Express 可以使用多种验证方案，其中既包括了内部验证方案，也包括了外部验证方案。
- 授权方案：授权方案也被称为可重用的访问控制规则，它们被统一地定义之后反复地应用于应用程序中的多个元素。例如，使用单一的授权方案，用户可以控制对一个字段的访问、对一个按钮部件的访问或对整个应用程序的访问。

（4）可以在应用程序中使用如下的 Oracle Application Express 互联网功能。

- 声明式开发：通过使用 Express 所提供的声明环境（包括向导和模板）来装配复杂的数据库驱动的互联网应用程序。
- 自动的会话状态管理：Oracle Application Express 在数据库中维护会话的状态，并赋予用户获取和设置应用程序中的任何页面的会话状态值的能力。在 Express 中每一个会话都被赋予一个唯一的标识符，而 Oracle Application Express 引擎正是利用这一会话 ID 来存储和检索应用程序的数据集。

除了以上介绍的这两种强大的功能之外，Oracle Application Express 还包括了一些其他互联网功能。

18.4　适合于使用Express开发的系统

可能会有读者问什么样的应用系统最适用于使用 Express 开发。Express 适用于开发以数据库为中心的应用系统，这些系统可能由分布在不同地点的开发小组开发，或者是它们的订货至交货时间很短，或者系统不是太复杂。总之 Express 适合于如下应用系统的开发：

- 基于互联网的项目追踪、合同追踪、租约追踪和资产追踪应用系统。
- 查找人、目录名等的应用系统。
- 带有报表和简单图表的不太重要的（轻量级的）商业智能应用系统。
- 使用正文索引和 Oracle 数据库搜寻能力的基于互联网的应用系统。
- 必须在很短的时间内创建的应用系统，通常在一周内，这些应用系统的订货至交货时间很短，同样它们的生命周期也很短。例如，事件注册应用系统、反馈表单系统、市场调查系统和民意调查系统，这些应用系统仅在某一特定的时间内使用，以后就完全没用了。

除此之外，由于 Express 是存在于 Oracle 数据库之中的，所以通过使用 PL/SQL 程序设计语言 Express 也可以开发出相当复杂的应用系统。

18.5　HTTP 服务器的选择和软硬件要求

尽管到目前为止，Oracle 公司提供了 3 种类型的 HTTP 服务器支持 Oracle Application Express，但是，为了降低读者学习的难度，本书采用其中最简单、也是最新的一种类型，就是嵌入式 PL/SQL 网关/连接器（Gateway）。嵌入式 PL/SQL 网关是安装在 Oracle 11g 数据库之中的，并且提供了带有互联网服务器和创建动态应用程序所必需的基本结构。嵌入式 PL/SQL 网关是运行在 Oracle 数据库的 Oracle XML DB 的 HTTP 服务器之中的。图 18.4 是它的体系结构示意图。

从图 18.4 可以清楚地看出，嵌入式 PL/SQL 网关是一个简单的两层体系结构，它由网络浏览器和 Oracle 数据库所组成，其中，嵌入式 PL/SQL 网关和 Oracle Application Express 包含在 Oracle 数据库之中。嵌入式 PL/SQL 网关的好处是：容易配置、包括在数据库之中、不需安装单独的服务器。

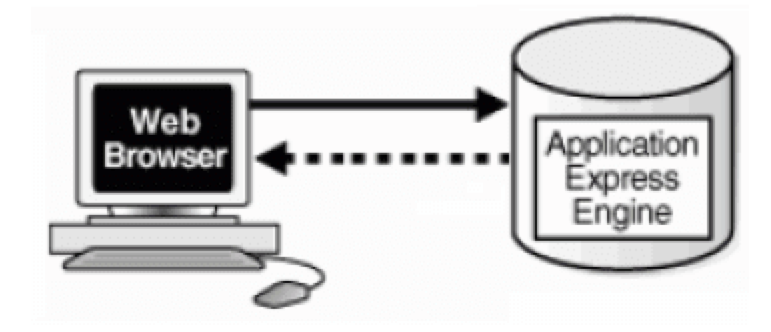


图　18.4

下面介绍安装 Oracle Application Express 的软件需求。在本书中使用的是 Oracle Application Express 的 3.2 版，它是 2009 年 2 月推出的。由于在本书中使用的是嵌入式 PL/SQL 网关，因此，需要使用 Oracle 11g 的数据库管理系统。如果是读者自己安装的数据

库，您应该清楚所安装的 Oracle 数据库的版本。如果是其他人安装的，而您又无法确定安装的版本时，可以使用数据字典 v$instance 或 v$version 来获取版本的信息。首先，使用例 18-1 的命令在 DOS 系统提示下，以 system 用户身份登录 Oracle 系统，wuda 是密码，在您的系统上可能不同。

例 18-1

```
sqlplus system/wuda
```

然后，使用例 18-2 的 SQL 语句从数据字典 v$instance 中获取数据库的实例名和版本号。

例 18-2

```
SQL> select instance_name, version from v$instance;
```

例 18-2 结果

```
INSTANCE_NAME    VERSION
---------------- ---------
moon             11.1.0.6.0
```

或者使用例 18-3 的 SQL 语句从数据字典 v$version 中获取版本的详细信息。

例 18-3

```
SQL> select * from v$version;
```

例 18-3 结果

```
BANNER
----------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
PL/SQL Release 11.1.0.6.0 - Production
CORE    11.1.0.6.0      Production
TNS for 32-bit Windows: Version 11.1.0.6.0 - Production
NLSRTL Version 11.1.0.6.0 - Production
```

☞ **指点迷津：**

如果读者安装的不是 Oracle 11g，不用卸载现有的 Oracle 数据库管理系统，只需将 Oracle 的所有服务改为手动，并停止它们，然后就可以安装 Oracle 11g 了。这样在使用 Oracle Application Express 时启动 Oracle 11g，而在进行以前的数据库操作时还可以使用之前版本的 Oracle 数据库，但是最好不要同时运行两个数据库。随书的光盘中有 Oracle 11g 的安装视频，另外，还包括了 Oracle 11g 的卸载视频。如果读者在安装 Oracle 11g 系统时遇到问题可以参考。

然后，要确认 Oracle 所使用的参数文件是否为二进制的参数文件，使用例 18-4 的 SQL*Plus 命令就能获得这一信息。

例 18-4

```
SQL> show parameter pfile
```

例 18-4 结果

```
NAME                                 TYPE        VALUE
------------------------------------ -----------
spfile                               string   F:\APP\ADMINISTRATOR\PRODUCT\1
                                              1.1.0\DB_1\DATABASE\SPFILEMOON
                                              .ORA
```

例 18-4 的显示结果中的 NAME 为 spfile，这就表明这个 Oracle 数据库使用的是二进制的参数文件（Oracle 11g 默认安装使用的是二进制参数文件）。

接下来要确认 Oracle 是否使用的是自动内存管理，使用例 18-5 的 SQL*Plus 命令就能获得这一信息。

例 18-5

```
SQL> show parameter memory_target
```

例 18-5 结果

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- -----
memory_target                        big integer 820M
```

从例 18-5 的显示结果可以看出，该数据库使用的是自动内存管理，因为 memory_target 的值为 820MB（不为 0），如果它为 0，即表明数据库使用的是手动内存管理，此时，需要将共享池的大小（shared_pool_size）加大到至少为 100MB。Oracle 11g 的默认安装是自动内存管理。

接下来要检查网络浏览器，Oracle Application Express 要求微软的网络浏览器为 6.0 或以上的版本，Firefox 为 1.0 或以上的版本。下面是检查微软的网络浏览器版本的具体操作：

（1）启动网络浏览器，如图 18.5 所示。



图　18.5

（2）选择“帮助”→“关于 Internet Explorer”命令，如图 18.6 所示。

（3）打开如图 18.7 所示的对话框。



图 18.6



图 18.7

从图 18.7 可知，所使用的微软网络浏览器的版本满足 Oracle Application Express 的安装要求。

安装 Oracle Application Express 所需的内存和磁盘空间如下：

- 物理内存最好为 2GB 或以上。
- Oracle Application Express 软件本身在操作系统的文件系统上需要至少 450MB 的空闲磁盘空间。
- Oracle 的系统（SYSTEM）表空间需要至少 85MB 的空闲磁盘空间。
- Oracle Application Express 所在的表空间需要至少 125MB 的空闲磁盘空间。
- 每安装一种其他语言（非英语），Oracle Application Express 所在的表空间需要增加至少 35MB 的额外空闲磁盘空间。

因为 Oracle 11g 默认安装的所有表空间的磁盘空间都可以自动扩展，所以为了减少初学者的学习难度，这里就不调整表空间的大小了。读者将来学习 Oracle DBA 的课程之后，就可以很轻松地调整表空间的大小。可使用例 18-6 的查询语句从数据字典 dba_data_files 中获取哪些表空间的磁盘空间可以自动扩展。

例 18-6

```
SQL> select tablespace_name, autoextensible from dba_data_files;
```

例 18-6 结果

```
TABLESPACE_NAME                AUT
------------------------------ ---
USERS                          YES
UNDOTBS1                       YES
SYSAUX                         YES
```

```
SYSTEM                          YES
EXAMPLE                         YES
```

例 18-6 的显示结果清楚地表明，这个 Oracle 数据库的所有表空间都可以自动扩展，因为所有表空间所对应的 autoextensible 列都为 YES。

在安装 Oracle Application Express 之前必须先获取 Oracle Application Express 的安装软件包，这里使用的是嵌入式 PL/SQL 网关的 3.2 版，其他版本的操作基本相同。该软件包可以在 Oracle 的官方网站 http://www.oracle.com/technology/products/database/application_express/download.html 上免费下载，个人用户可以免费无限期地使用 Oracle Application Express。

18.6　Oracle Application Express安装

做完了以上的准备工作之后，就可以开始安装和配置 Oracle Application Express 了。以下是使用嵌入式 PL/SQL 网关/连接器（Gateway）安装和配置 Oracle Application Express 的具体步骤：

（1）安装 Oracle Application Express。

（2）修改 ADMIN 账户的密码。

（3）配置嵌入式 PL/SQL 网关。

（4）核实和开启 Oracle XML DB HTTP 服务器的端口。

（5）开启 Oracle 11g 数据库中的网络服务。

（6）安装其他语言。

（7）设置 JOB_QUEUE_PROCESSES 参数。

（8）配置 SHARED_SERVERS 参数。

接下来就开始按顺序详细地介绍每一步骤。在安装之前要先解压缩下载的 Oracle Application Express 软件包，并将其复制到指定的安装目录下，如图 18.8 所示。

提示：

为了方便读者，在随书光盘的 ch18 目录中有一个叫 Install.txt 的文本文件，其中按顺序存放了以下安装和配置 Oracle Application Express 所需的全部命令，读者复制相应的命令之后粘贴到 SQL*Plus 中直接运行即可。另外，读者在安装时如果遇到问题，也可以参考光盘中的 Oracle Application Express 安装和配置的教学视频。

当确认Oracle Application Express软件包已经复制到指定目录之后，就可以按如下的步骤进行安装了：

（1）安装 Oracle Application Express。

① 启动 DOS 窗口，并使用 cd 命令（F:\ >cd app\apex）将当前目录切换到 Express 软件包所在的目录 F:\app\apex（您的系统上可能是不同的目录），如图 18.9 所示。



图 18.8　　　　图 18.9

② 使用/nolog 选项启动 SQL*Plus，如例 18-7 所示。

例 18-7

```
F:\app\apex>sqlplus /nolog
```

例 18-7 结果

```
SQL*Plus: Release 11.1.0.6.0 - Production on 星期二 5 月 12 08:34:45 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
SQL>
```

③ 使用例 18-8 的 SQL*Plus 命令（wuda 为密码），以 SYS 用户身份登录 Oracle 数据库管理系统。

例 18-8

```
SQL> connect sys/wuda as sysdba
```

例 18-8 结果

```
已连接。
SQL>
```

④ 使用例 18-9 的 SQL*Plus 命令运行安装脚本 apexins.sql 文件进行全部开发环境的安装，如图 18.10 所示。

例 18-9

```
@apexins SYSAUX SYSAUX TEMP /i/
```

在这里首先介绍一下例 18-9 的 SQL*Plus 命令中各个选项的含义，apexins 为安装脚本文件名，两个 SYSAUX 表示 Express 的数据和用户信息都将存放在 SYSAUX 表空间中，TEMP 为排序所用的表空间，“/i/”表示 Express 中存放图像的虚拟目录。安装会持续一段时间，等出现了如图 18.11 所示的界面时，就表示安装已经成功。

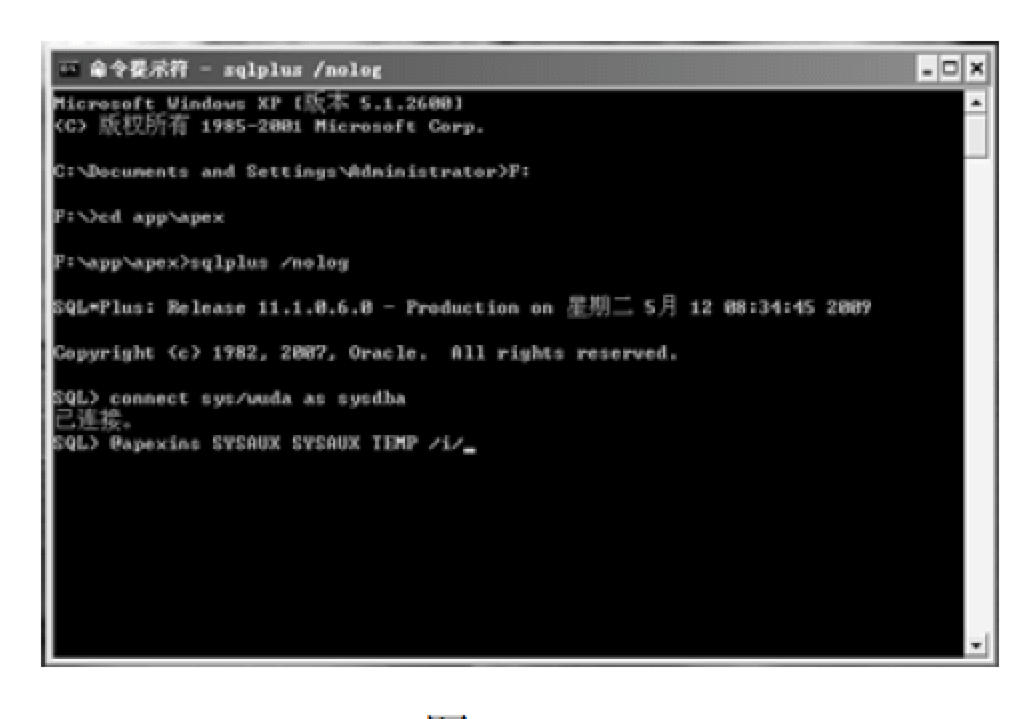


图　18.10



图　18.11

当 Oracle Application Express 安装成功之后，Express 会在 Oracle 11g 数据库中创建如下 3 个新的账户（用户）。

- APEX_030200：该账户拥有 Oracle Application Express 模式和元数据。
- FLOWS_FILES：该账户拥有 Oracle Application Express 上传的文件。
- APEX_PUBLIC_USER：该账户只拥有配置 Oracle HTTP 服务器和 mod_plsql 网关的 Oracle Application Express 的最小权限。

那么，如何确定 Express 已经成功地创建了这 3 个用户呢？由于当 Oracle Application Express 安装完成之后，它会自动地退出 SQL*Plus，所以首先必须以 SYS 用户重新登录数据库系统，然后使用例 18-10 和例 18-11 的查询语句从数据字典 DBA_USERS 中获取这方面的信息。

例 18-10

```
SQL> select username, created
  2  from dba_users
  3  where username like 'APEX%';
```

例 18-10 结果

```
USERNAME                        CREATED
------------------------------ ----------
APEX_030200                     12-5 月 -09
APEX_PUBLIC_USER                15-10 月-07
```

例 18-11

```
SQL> select username, created
  2  from dba_users
  3  where username like 'FLOW%';
```

例 18-11 结果

```
USERNAME                        CREATED
------------------------------ ----------
FLOWS_FILES                     15-10 月-07
FLOWS_030000                    15-10 月-07
```

当确认了 Oracle Application Express 确实创建了所需的用户之后，就可以修改 ADMIN

账户的密码了（ADMIN 为 Express 的系统管理员用户，是自动安装的）。

（2）修改 ADMIN 账户的密码。

在 SYS 用户中输入例 18-12 的 SQL*Plus 命令运行修改 ADMIN 账户密码的脚本文件 apxchpwd.sql，在提示处输入 ADMIN 账户密码，这里输入的是 wuda（武大），如图 18.12 所示。

例 18-12

```
SQL> @apxchpwd
```

（3）配置嵌入式 PL/SQL 网关。

① 输入例 18-13 的 SQL*Plus 命令运行配置脚本文件 apex_epg_config.sql，其中 F:\app 为 Oracle Application Express 的安装目录，如图 18.13 所示。

例 18-13

```
@apex_epg_config F:\app
```



图 18.12



图 18.13

② 输入例 18-14 的 SQL 语句将 ANONYMOUS 账户解锁。

例 18-14

```
SQL> ALTER USER ANONYMOUS ACCOUNT UNLOCK;
```

例 18-14 结果

用户已更改。

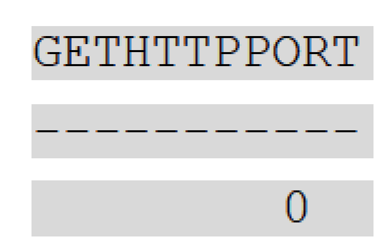
（4）核实 Oracle DB HTTP 服务器的端口。

① 输入例 18-15 的 SQL 语句核实 Oracle XML DB HTTP 服务器的端口。

例 18-15

```
SQL> SELECT DBMS_XDB.GETHTTPPORT FROM DUAL;
```

例 18-15 结果



```
GETHTTPPORT
-----------
          0
```

因为端口号为 0，所以 Oracle XML DB HTTP 服务器是关闭的。

② 输入例 18-16 的 SQL 语句开启 Oracle XML DB HTTP 服务器（端口号可以自选，不一定是 8080。但是最好不要选 1024 以下的端口号，因为它们中的一些可能已经被操作系统使用了）。

例 18-16

```
SQL> EXEC DBMS_XDB.SETHTTPPORT(8080);
```

例 18-16 结果

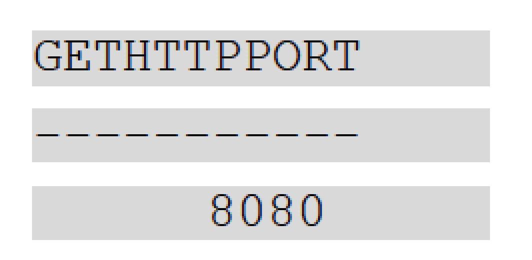
```
PL/SQL 过程已成功完成。
```

然后，再使用与例 18-15 完全相同的 SQL 语句例 18-17 查看所设置的端口号。

例 18-17

```
SQL> SELECT DBMS_XDB.GETHTTPPORT FROM DUAL;
```

例 18-17 结果



```
GETHTTPPORT
-----------
       8080
```

例 18-17 的显示结果表明 Oracle XML DB HTTP 服务器已经开启，其端口号为 8080。

（5）运行图 18.14 中的 PL/SQL 语句来开启 Oracle 11g 数据库中的网络服务（读者不用理解这段程序，随书光盘中有一个叫 service.sql 的文件，读者只要复制之后粘贴到 SQL*Plus 中运行即可）。

到此为止，您已经可以使用网络浏览器连接到 Oracle Application Express 了，只不过显示的界面都是英文的。启动网络浏览器并在地址栏中输入 http://localhost:8080/apex/就可以连接到 Oracle Application Express 管理服务程序，如图 18.15 所示。



图　18.14

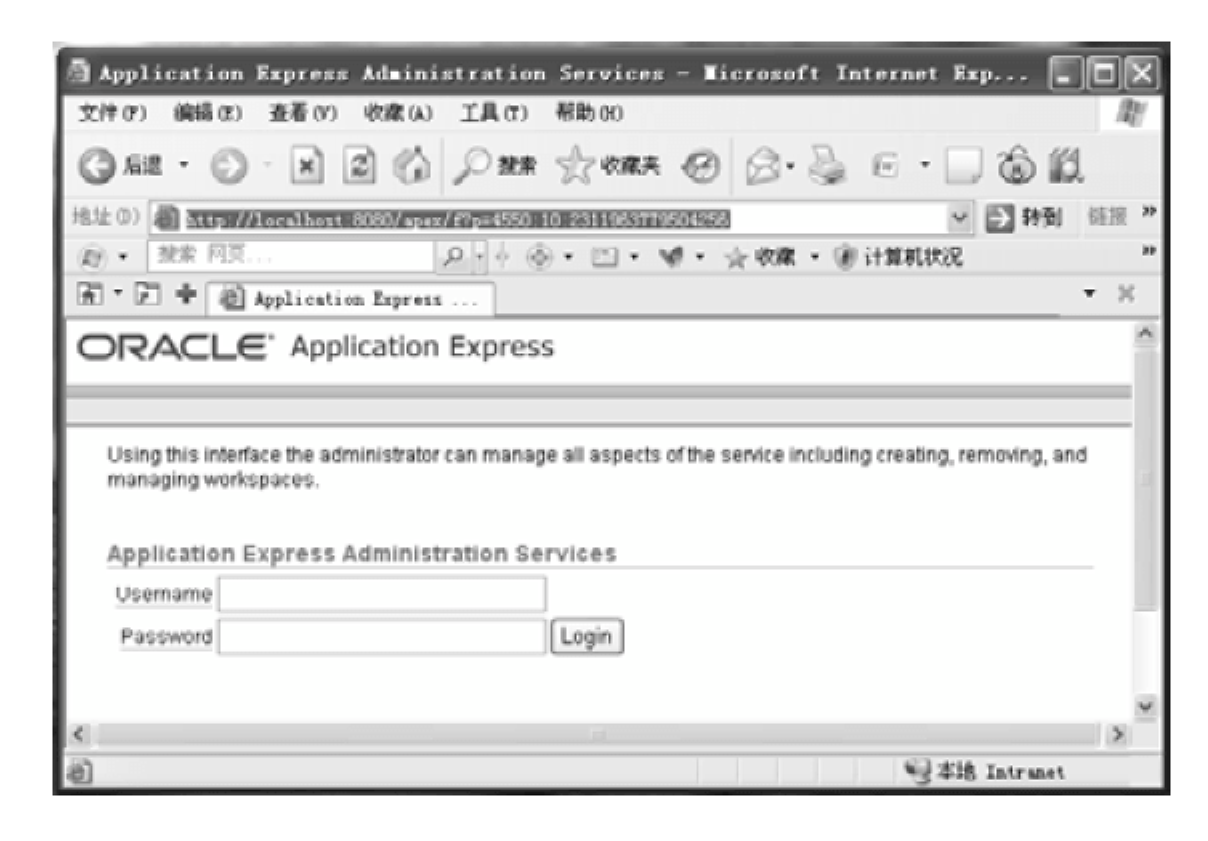


图　18.15

（6）安装中文。

为了适应中国人的需要，下面将安装中华民族引以为自豪的、具有几千年光辉灿烂历

史、并成就了无数文人墨客的古老文字——汉字。

① 按如下的方式设置操作系统环境变量 NLS_LANG。

选择“开始”→“控制面板”命令，打开“控制面板”窗口，单击“性能和维护”超链接，双击“系统”图标，打开“系统属性”对话框，选择“高级”选项卡，单击“环境变量”按钮，打开“环境变量”对话框，单击“新建”按钮，打开“新建用户变量”对话框，在“变量名”文本框中输入“NLS_LANG”，在“变量值”文本框中输入“American_America. AL32UTF8”，然后一直单击“确定”按钮就完成了操作系统环境变量的设置，如图 18.16 所示。

② 启动 DOS 窗口，并以“f:”和 cd app\apex 命令切换到 Oracle Application Express 的安装目录。

③ 使用“sqlplus sys/wuda as sysdba”命令启动 SQL*Plus 并以 SYS 用户登录 Oracle 数据库系统，如图 18.17 所示。



图 18.16

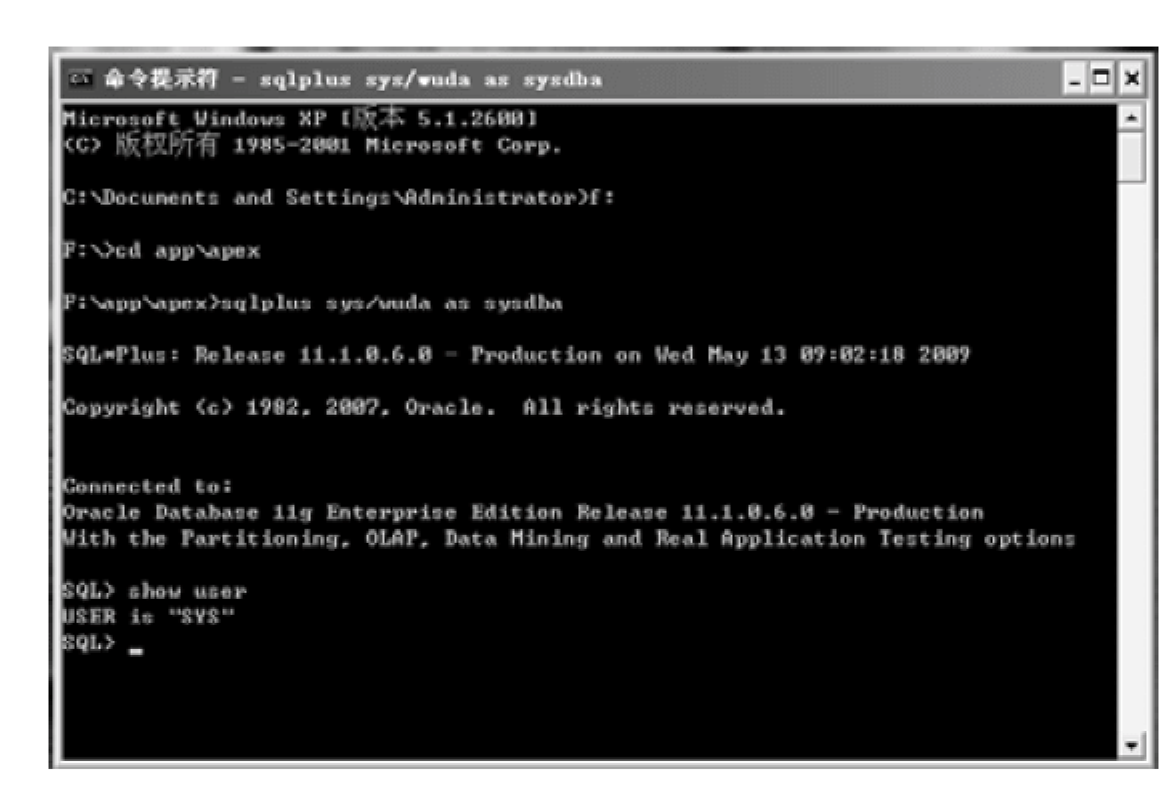


图 18.17

④ 执行例 18-18 的 SQL 语句将当前的会话用户切换为 APEX_030200。

例 18-18

```
ALTER SESSION SET CURRENT_SCHEMA = APEX_030200;
```

⑤ 运行相应语言（简体中文）的脚本文件。

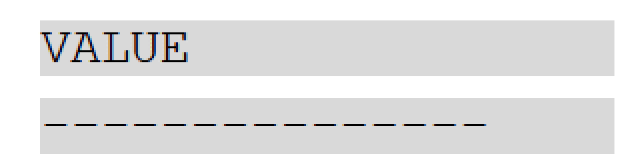
```
@F:\app\apex\builder\zh-cn\load_zh-cn.sql
```

（7）核实和设置 job_queue_processes 参数。要使 Oracle Application Express 正常工作，job_queue_processes 参数要设为至少 20 或以上。可以使用例 18-19 的查询语句从数据字典 v$parameter 中获取该参数的值。

例 18-19

```
SELECT VALUE FROM v$parameter WHERE NAME ='job_queue_processes'
```

例 18-19 结果



```
VALUE
---------------
```

```
1000

1 row selected.
```

也可以使用 SQL*Plus 的 show parameter job 命令来获取同样的信息，如例 18-20。

例 18-20

```
SQL> show parameter job
```

例 18-20 结果

```
NAME                                  TYPE                    VALUE
------------------------------------------------------------ -----
job_queue_processes                   integer                 1000
```

由于在我们的系统中该参数已经为 1000，足够大了，所以不需要再设置这一参数。否则需要使用以下的命令来设置这一参数。

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES = <数值>
```

（8）核实并设置 shared_servers 参数。要使 Oracle Application Express 正常工作，必须将 shared_servers 参数设置为大于 1 的数值。对于并行操作用户数量较少的系统，可以将这一参数设置为 5。

① 使用例 18-21 的 SQL*Plus 命令查看当前 shared_servers 参数的值。

例 18-21

```
SQL> show parameter shared_servers
```

例 18-21 结果

```
NAME                           TYPE                               VALUE
------------------------------ ---------------------------------- -----
max_shared_servers             integer
shared_servers                 integer                              1
```

例 18-21 的显示结果表明，shared_servers 参数的当前值为 1，因此需要重新设置。

② 使用例 18-22 的 ALTER SYSTEM SET 命令将 shared_servers 参数的值设置为 5。

例 18-22

```
SQL> ALTER SYSTEM SET SHARED_SERVERS = 5 SCOPE=BOTH;
```

例 18-22 结果

```
System altered.
```

例 18-22 命令的 SCOPE=BOTH 选项表示系统要同时修改内存和二进制初始化参数文件中的 shared_servers 参数的值。然后再使用与例 18-21 完全相同的 SQL*Plus 命令例 18-23 确认 shared_servers 参数的值是否已经为 5 了。

例 18-23

```
SQL> show parameter shared_servers
```

例 18-23 结果

NAME	TYPE	VALUE
----------------------------	----------------------------------	-----
max_shared_servers	integer	
shared_servers	integer	5

到此为止，我们终于完成了全部的 Oracle Application Express 的安装和配置。接下来就可以创建 Express 用户使用的工作区和在工作区中添加 Express 用户了。

18.7 Express工作区和用户角色

首先介绍一下什么是工作区（Workspace），简单地说，工作区就是用户用来开发应用系统的地方。工作区是一个虚拟的私有数据库，它能使多个用户工作在相同的 Oracle Application Express 系统的同时，还能保持它们的对象、数据和应用系统为私有。每一个工作区都有一个唯一的 ID 和名字。

在一个典型的开发环境中，可能只创建了一个为所有开发人员共享的工作区。当然，也可以为某些开发人员或项目创建专用的工作区。这样可以限制对工作区的访问，只允许那些与工作区（或项目）相关的人员访问特定的工作区。图 18.18 显示了用户和开发人员、工作区以及数据库模式（用户）之间的关系。

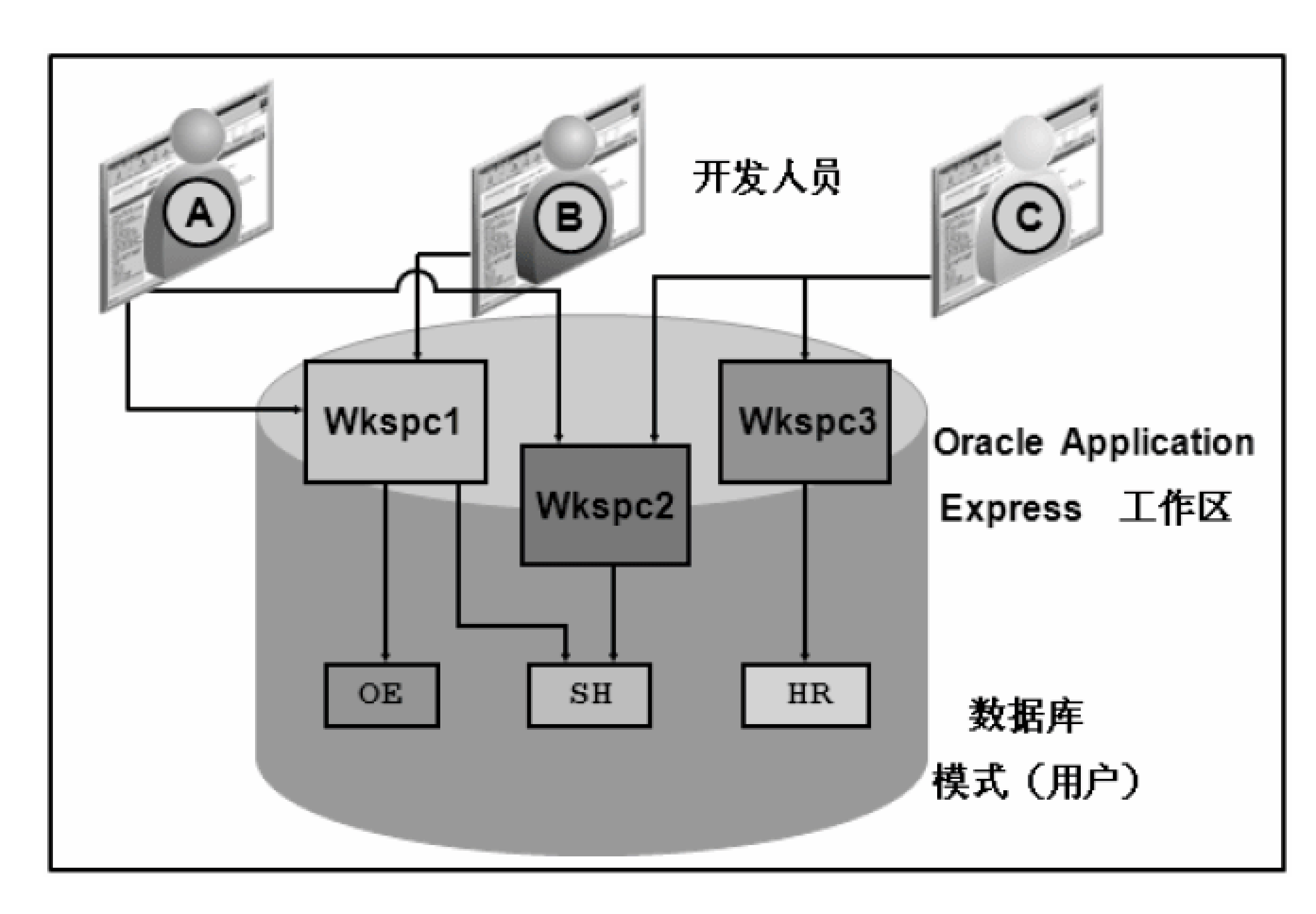


图 18.18

当一个用户登录 Oracle Application Express 时，他所访问的是 Express 开发环境中的一个工作区。一个工作区中可以有一个或多个开发人员，而用户（开发人员）是在他的工作区上开发应用系统的，一个工作区又松散地对应着一个或多个数据库模式。因此，当一个开发人员在开发应用系统时，他就可以使用在他的工作区中可以获得的数据库模式中的数据库对象。

如图 18.18 所示，一个单一的 Oracle 数据库可以包含多个 Oracle Application Express 工作区。在这个图中，A、B 和 C 为开发人员；Wkspc1、Wkspc2 和 Wkspc3 是 3 个不同的工

作区。A 和 B 可以访问 Wkspc1，A 还可以访问 Wkspc2，C 可以访问 Wkspc2 和 Wkspc3。每一个工作区可以访问一个或多个数据库模式（用户）。例如，Wkspc1 可以访问模式（用户）OE 和 SH，Wkspc2 可以访问模式 SH，而 Wkspc3 可以访问模式 HR。因此，多个开发人员既可以利用不同的工作区，也可以利用相同的工作区在同一个数据库实例上工作，而且这些工作区既可以访问相同的模式也可以访问不同的模式。

最终，Oracle Application Express 将一个单一的 Oracle 数据库转换成一个共享的工作组数据库服务。这个服务可以通过网络浏览器访问，而且在开发人员和用户的 PC 或工作站上不需要安装。

接下来我们简单地介绍一下什么是模式。模式是一个逻辑上包含数据库对象的容器，其对象如表、视图和存储过程等。一个单一的模式可以与一个或多个工作区相关。

Oracle Application Express 为了安全和方便管理将用户划分为如下 4 种不同的类型(角色)。

- Oracle Application Express 管理员：为超级用户，他管理整个 Oracle Application Express 实例，其中既包括服务的管理，也包括工作区的管理，默认的用户名为 admin。
- Oracle Application Express 开发人员：可以创建和编辑应用系统的用户，开发人员可以有自己的工作区，也可以与其他的开发人员共享一个工作区。
- Oracle Application Express 工作区管理员：当一个开发人员被授予了对他的工作区的管理权限时，他就成为了工作区管理员。工作区管理员可以在他的工作区中添加新的用户、创建新的用户组和查看他的工作区的使用报告。
- Oracle Application Express 终端用户：没有开发和管理权限的用户，他只有运行应用程序的基本权限。

18.8　设置自己的本地环境

本章只介绍一种简单的设置用户开发环境的方法，其目的是为了使读者尽快地学会使用 Oracle Application Express。如果读者想要设置适合于生产需要的开发环境，可以参阅文档 Oracle Application Express Administration Guide。该文档可以从 Oracle 公司的官方网站上免费下载。设置的具体步骤如下：

（1）登录 Oracle Application Express 管理服务。

Oracle Application Express 管理服务是一个单独的应用程序，它被用来管理整个的 Oracle Application Express 实例。登录的方法如下。

① 启动网络浏览器，在地址栏输入“http://主机名:端口号/apex/apex_admin”，单击“转到”按钮，其中，

- 主机名。是安装 Oracle Application Express 服务器的计算机的名字，这里可以使用 localhost，即本地主机。
- 端口号。是赋予 Oracle Application Express 服务器的端口号，默认为 8080。
- Apex。是在配置文件中定义的数据库访问描述符（DAD）。如果用户是从早期版

本升级过来的或有人配置过，数据库访问描述符可能是 htmldb 或其他。有关这方面的内容读者可以参阅文档 Oracle Application Express Builder User's Guide，该文档可以从 Oracle 公司的官方网站上免费下载。

现在启动网络浏览器，在地址栏输入“http://localhost:8080/apex/apex_admin”，单击“转到”按钮即可打开如图 18.19 所示的登录页面。

② 在“用户名”文本框中输入 admin。

③ 在“口令”文本框中输入密码，这里是 wuda（武大）。

④ 单击“登录”按钮。如果是第 1 次登录，Express 会出现如图 18.20 所示的页面，要求用户修改口令。

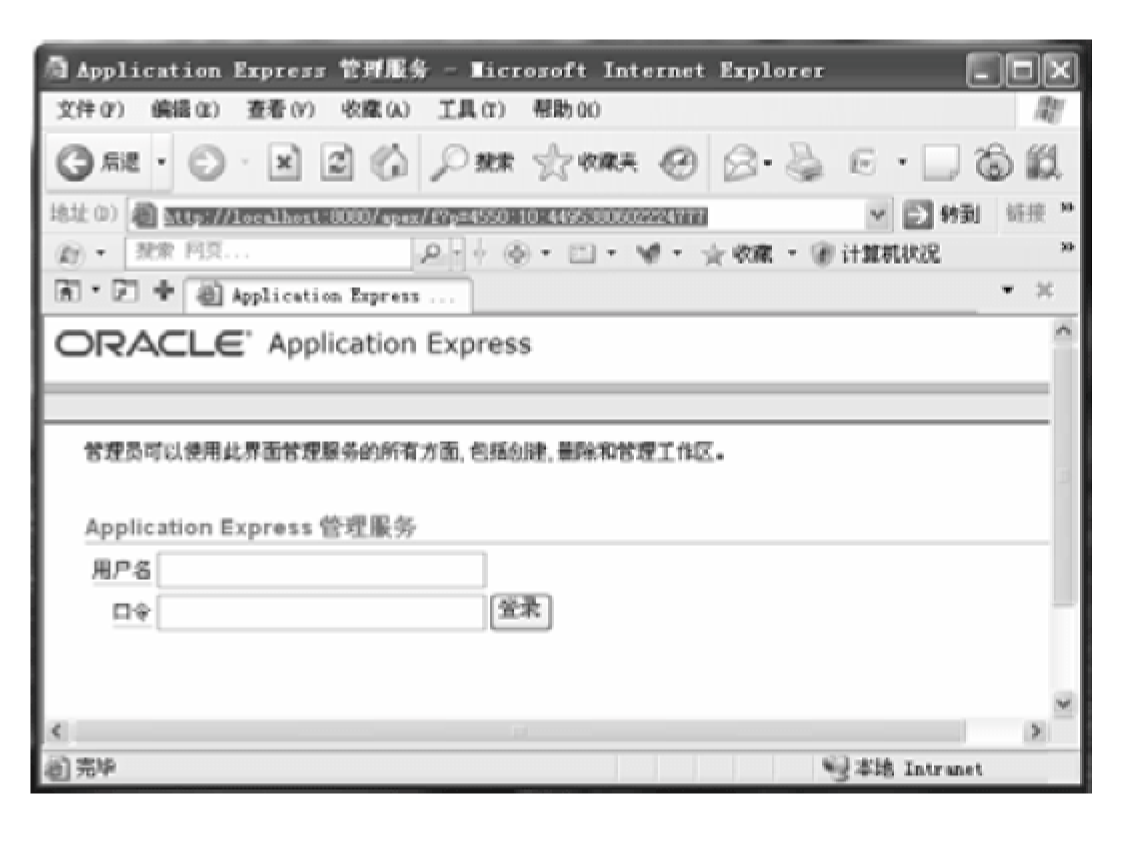


图 18.19



图 18.20

在“请输入当前口令”文本框中输入原来的密码（wuda），在“请输入新口令”文本框中输入新的密码，这里输入“xm_Q1ng”（西门庆），在“请确认新口令”文本框中继续输入“xm_Q1ng”，单击“应用更改”按钮，接下来再单击“返回”按钮，就会打开与图 18.19 相同的页面，此时使用新的密码 xm_Q1ng 登录就会得到如图 18.21 所示的页面。然后就可以创建工作区了。

（2）创建工作区。

① 单击“管理工作区”图标，将出现如图 18.22 所示的页面。




图 18.21



图 18.22

② 在“管理工作区”图标右侧单击 “创建工作区”超链接，会打开如图 18.23 所示的页面。

③ 在“工作区名称”文本框中输入“hr”，在“工作区说明”列表框中可以输入任何描述性的信息，然后单击“下一步”按钮，就会打开如图 18.24 所示的页面。

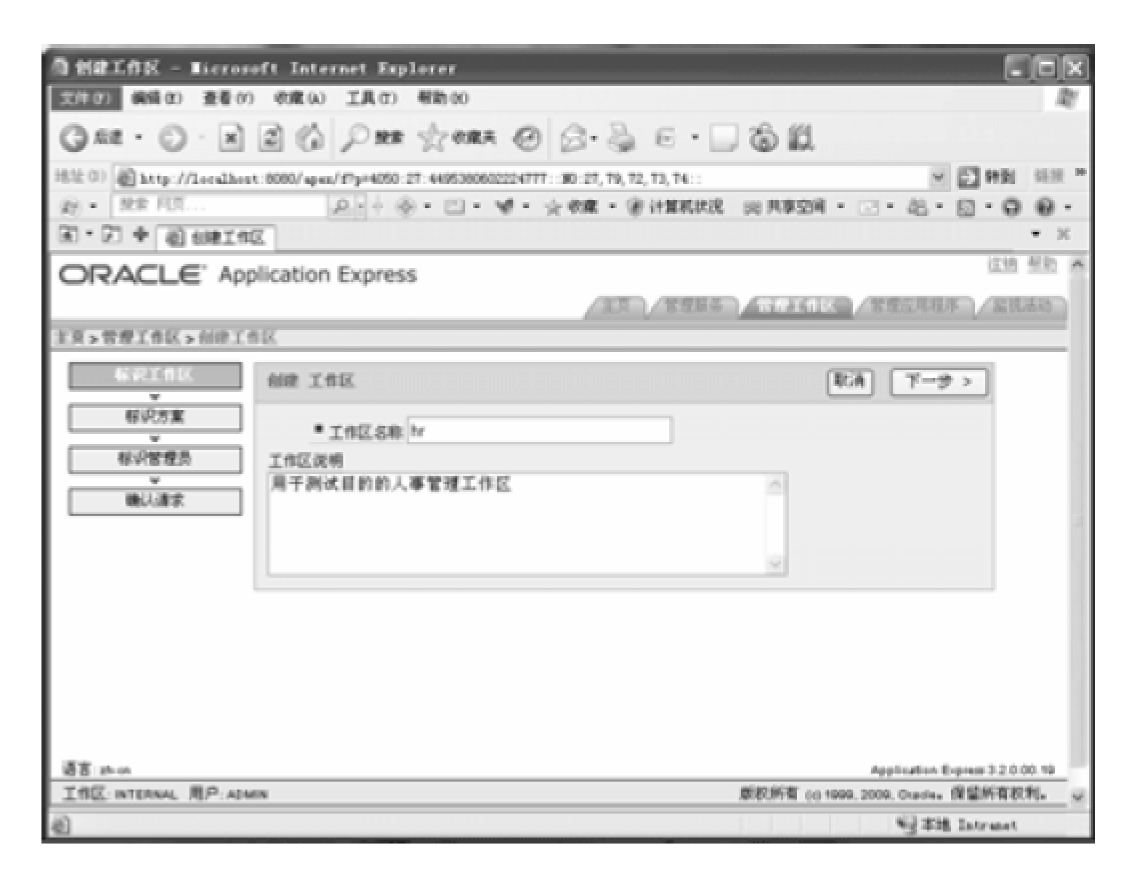


图　18.23

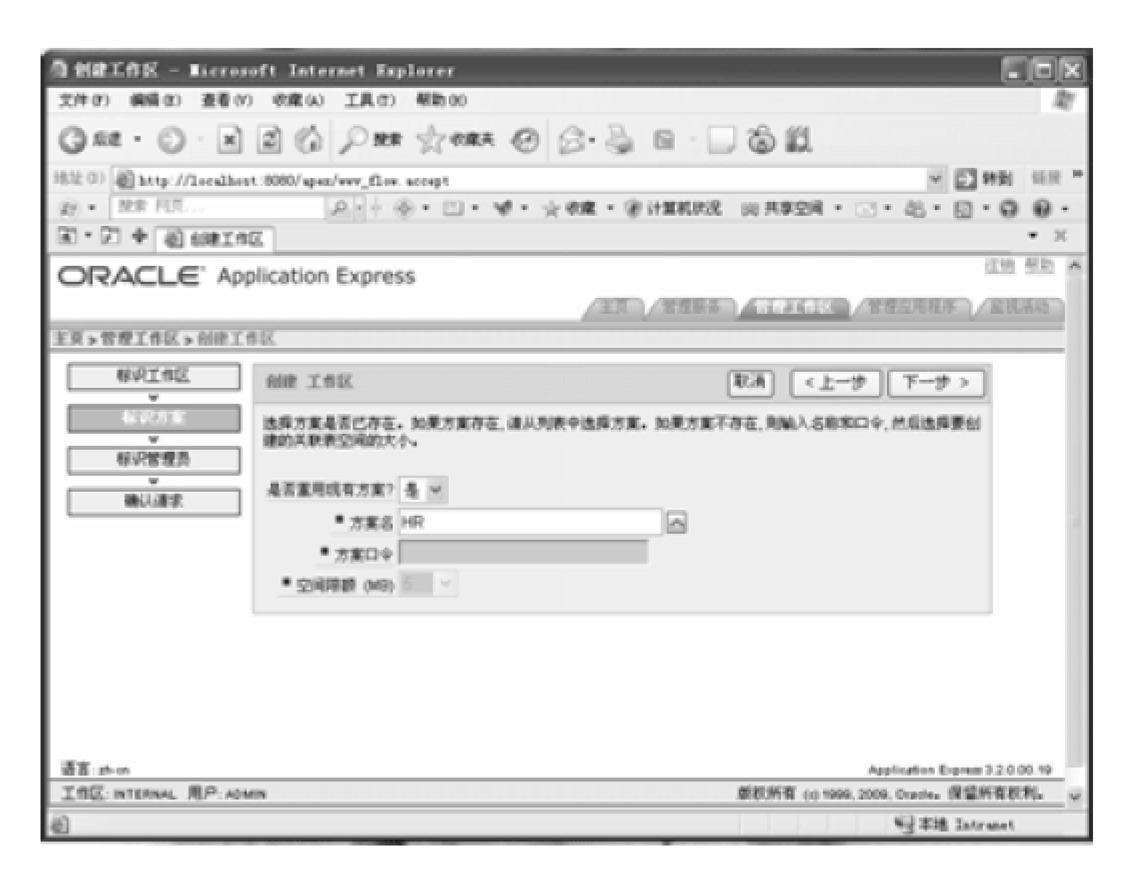


图　18.24

④ 在“是否重用现有方案”下拉列表框中选择“是”选项，在“方案名”下拉列表框中选择 HR 选项，然后单击“下一步”按钮，就会打开如图 18.25 所示的页面。

⑤ 在“管理员口令”文本框中输入密码“xm_Q1ng”，接着输入名、姓和电子邮件，然后单击“下一步”按钮，就会打开如图 18.26 所示的页面。

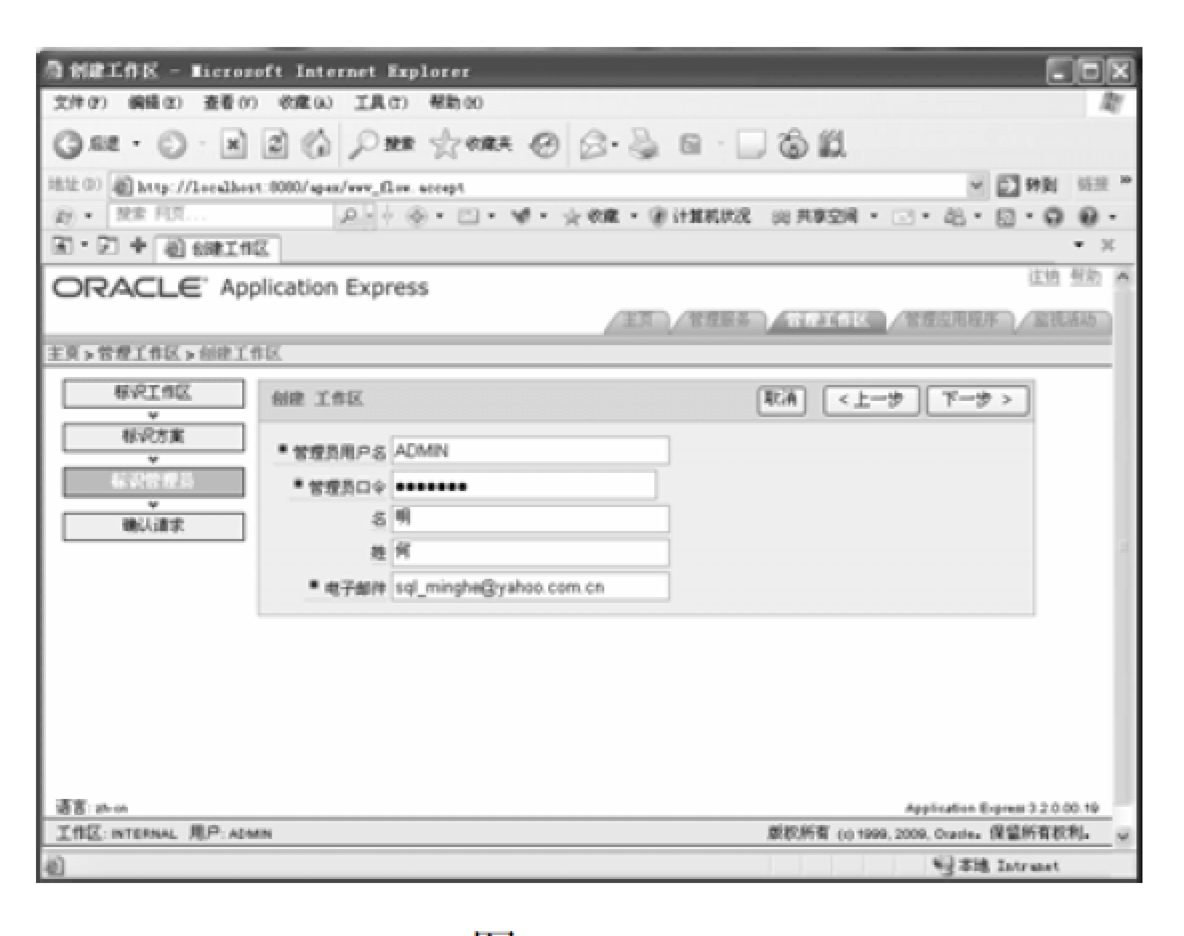


图　18.25

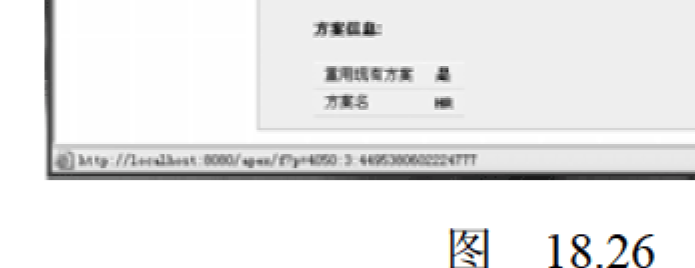


图　18.26

⑥ 此时，用户可以检查所设定的信息是否准确，如果没有问题，就单击“创建”按钮，打开如图 18.27 所示的页面。

⑦ 单击“完成”按钮，就回到如图 18.28 所示的管理工作区界面。

到此为止，我们已经成功地创建了一个名为 hr 的工作区。现在可以选择“文件”→“关闭”命令关闭管理工作区的窗口。

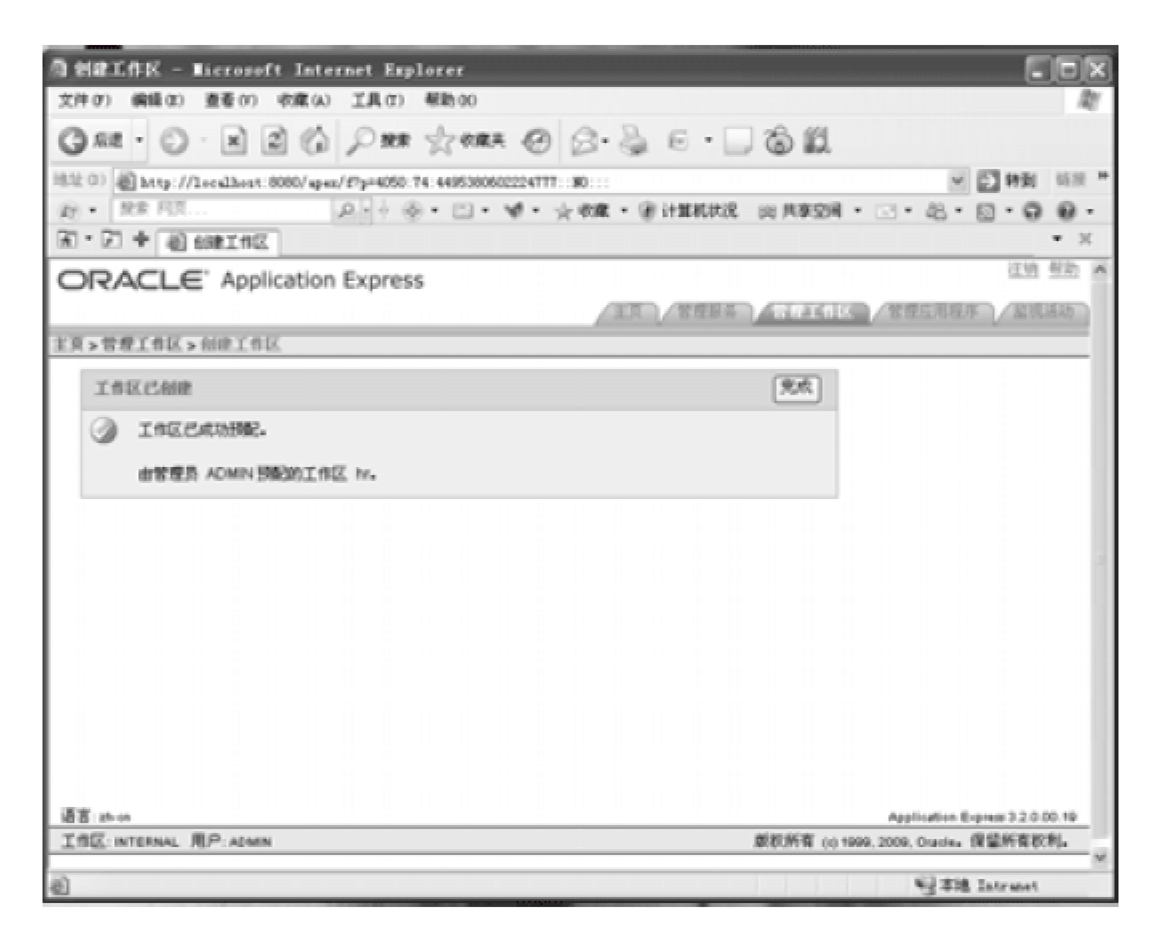


图 18.27



图 18.28

18.9　登录本地Express实例

要登录本地 Oracle Application Express 实例，用户必须提供如下的信息：

- 用户可以访问的 Oracle Application Express 网络地址（URL）。
- 用户用来开发应用系统的工作区的名字。
- 在 Oracle Application Express 中为该工作区所设立的用户名。
- 在 Oracle Application Express 中为该工作区所设立的密码。

如果是您自己设置的开发环境，您应该已经有了以上这些信息。否则的话，您就要从为您创建用户的人（Express 管理员）那里获取这些信息。登录 Oracle Application Express 的具体步骤如下：

（1）启动网络浏览器，在地址栏输入“http://localhost:8080/apex”，单击“转到”按钮。其中，localhost 为主机名，8080 为端口号，apex 为数据库访问描述符（DAD），打开如图 18.29 所示的页面。

（2）在“工作区”文本框中输入“hr”，在“用户名”文本框中输入“admin”，在“口令”文本框中输入“xm_Q1ng”，然后单击“登录”按钮，就会打开如图 18.30 所示的页面。

由于是以 ADMIN（Express 管理员）用户身份登录的，所以您将具有所有开发人员的权限。此时，如果您愿意的话可以马上开始开发应用系统。但是我们知道 ADMIN 是一个超级用户，他具有至高无上的权力，以这样的身份进行系统的开发和日常维护工作，风险相当大，因为一些失误可能导致灾难性的后果。因此，比较稳妥的方法是再创建一个或多个开发人员用户，使用这些用户进行应用系统的开发。

一般在系统管理中采用的一个原则是“最小化原则”，其原理是在能完成工作的前提下，使用权限最低的用户。这样万一有失误，对系统所造成的破坏最小。其实，现实生活中也是使用同样的原理，例如，氢弹是大家知道的最强大的杀伤武器，但是自从这种超级武器诞生以来，还没有哪个国家的领导人敢按下发射的按钮。

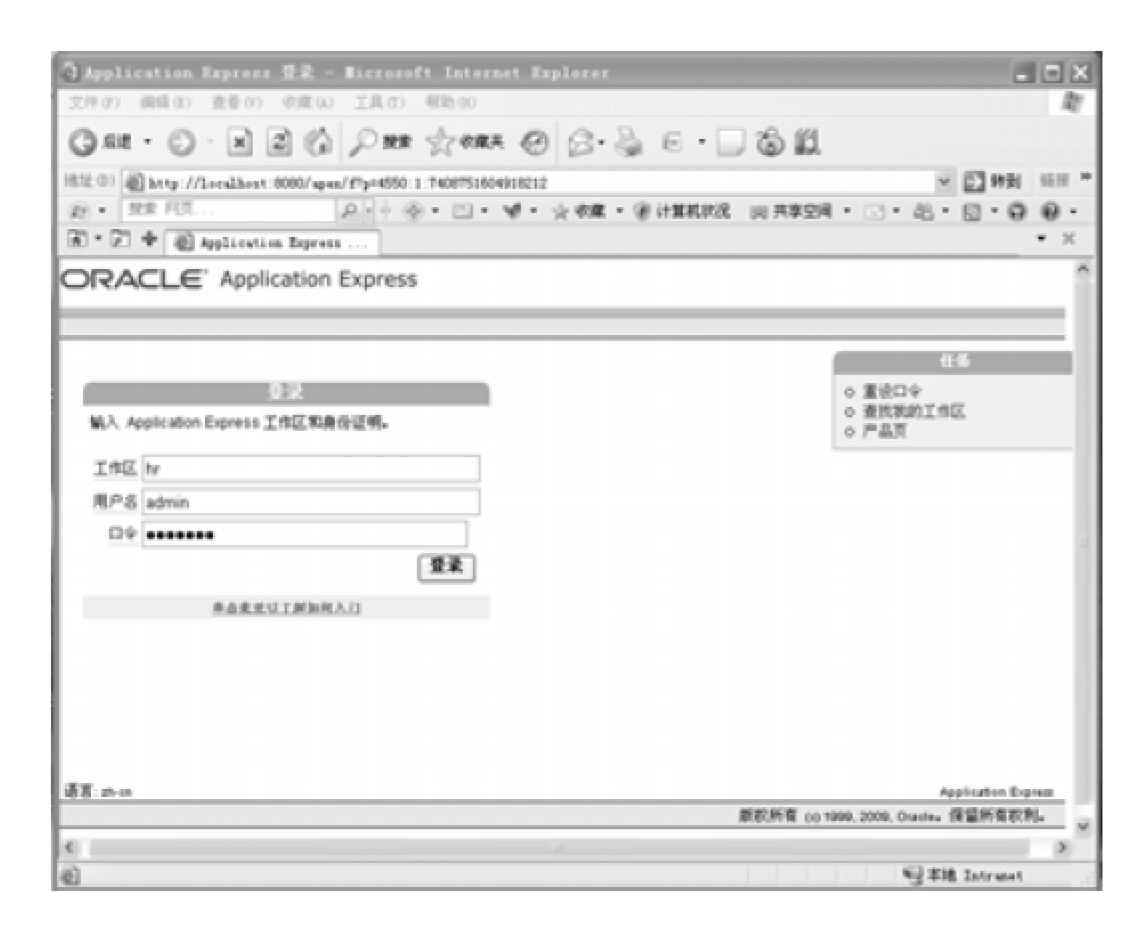


图　18.29



图　18.30

18.10　创建新用户（账户）

接下来我们将创建一个共享 HR 工作区的具有开发人员角色（权限）的新用户，用户名为 dog（狗），为了简单起见，密码还是 xm_Q1ng。以下就是具体的操作步骤：

（1） 在 Oracle Application Express 主页右侧的“管理”面板中单击“管理 Application Express 用户”超链接（如图 18.30 所示），然后打开如图 18.31 所示的页面。

（2）在右下方的“任务”列表中单击“创建开发者”超链接，就会出现如图 18.32 所示的创建用户的页面。



图　18.31

图　18.32

（3）在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”，在“确认口令”文本框中输入“xm_Q1ng”，输入用户的电子邮件地址，开发者权限部分保留为默认值，然后继续填写账户控制等信息，如图 18.33 所示。

（4）在“设置账户可用性”下拉列表框中选择“未锁定”选项，在“需要在首次使用时更改口令”下拉列表框中选择“否”选项，不必管用户组，在“附加属性”区域按您的兴趣输入。

（5）选择创建用户，就会得到如图 18.34 所示的页面。

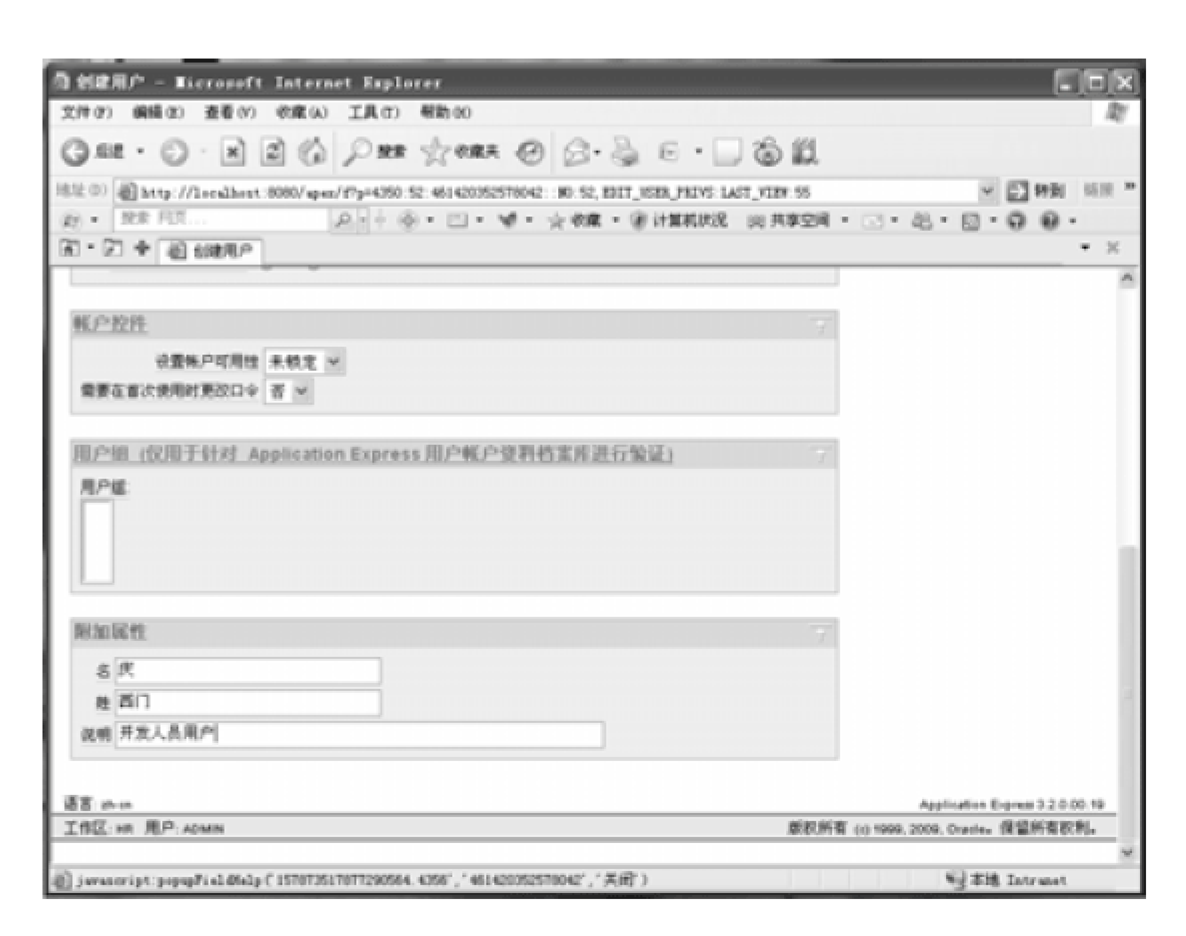


图　18.33



图　18.34

从图 18.34 中可以清楚地看出，我们已经成功地创建了用户 DOG。现在就可以选择“文件”→“关闭”命令退出 Oracle Application Express 了。

谢天谢地，读者现在终于可以长出一口气了，因为您已经读完了在 Oracle Application Express 学习过程中最令人望而生畏的一章。接下来将学习如何将您的数据库中的数据随心所欲地放在互联网上。

第19章

Express 的用户界面

在开始使用 Oracle Application Express 进行应用系统的开发之前，读者要先熟悉 Express 的用户界面和使用方法。为了讲解方便，我们首先要登录 Oracle Application Express。为此，启动网络浏览器并在地址栏中输入“http://localhost:8080/apex/”连接到 Oracle Application Express 管理服务程序，如图 19.1 所示。

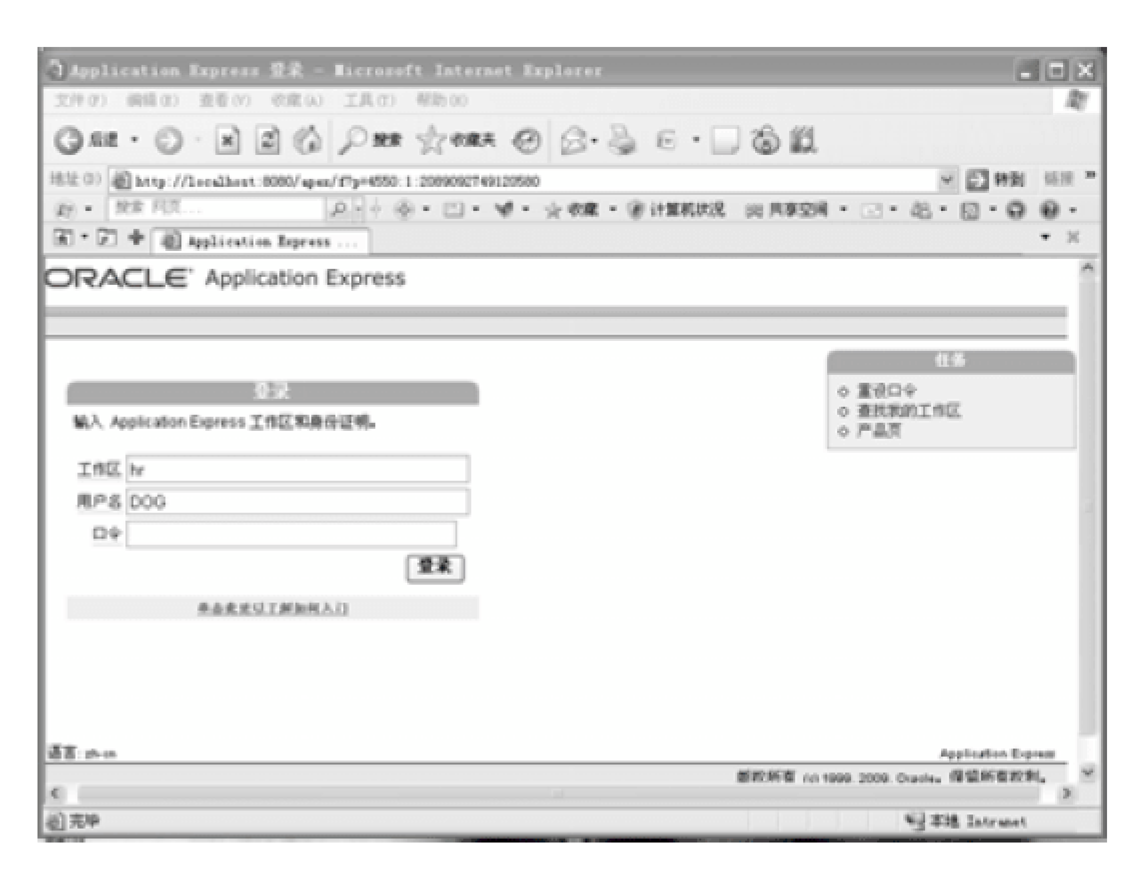


图 19.1

在登录页面的“工作区”文本框中输入“hr”、在“用户名”文本框中输入“admin”，在“口令”文本框中输入“xm_Q1ng”（是创建用户时设定的，可能不同），然后单击“登录”按钮即可登录 Express。

19.1 Express工作区主页

当登录了 Oracle Application Express 时，系统就将出现如图 19.2 所示的工作区主页。

注意您的工作区名和用户名会出现在工作区主页的左下角。在主页的中心部分有 3 个大的图标，它们分别是应用程序构建器、SQL 工作室和实用程序。接下来我们简单介绍每一个组件的功能。

- 应用程序构建器：用来在数据库对象（如表和过程）上组装 HTML 或应用程序。这也是开发人员使用最多的组件。
- SQL 工作室：是一个访问数据库的工具，可以用来查看和管理数据库对象。
- 实用程序（Utilities）：用来从数据库中导出或向数据库中导入数据、生成数据定

义语言（DDL）、查看对象报表、修复已经删除的数据库对象和执行其他任务。

1. “管理”列表

“管理”列表出现在工作区主页的右侧，利用该列表中的超链接来管理应用程序开发环境。如果用户具有管理员和开发人员权限，将会有如下的超链接，如图 19.3 所示。



图 19.2

管理

- 管理
  - 管理服务
  - 管理 Application Express 用户
  - 监视活动
- 更改口令
- 关于 Application Express

图 19.3

（1）管理

单击“管理”超链接用于查看管理任务列表，如果用户是开发人员，其权限是有限的。该列表包括如下的任务。

- 管理服务（以管理员身份登录）：单击该超链接可以用来管理会话状态、日志文件、工作区首选项和应用程序模型等，如图 19.4 所示。
- 管理 Application Express 用户（以管理员身份登录）：单击该超链接可以用来管理用户的账户和用户组，如图 19.5 所示。



图 19.4

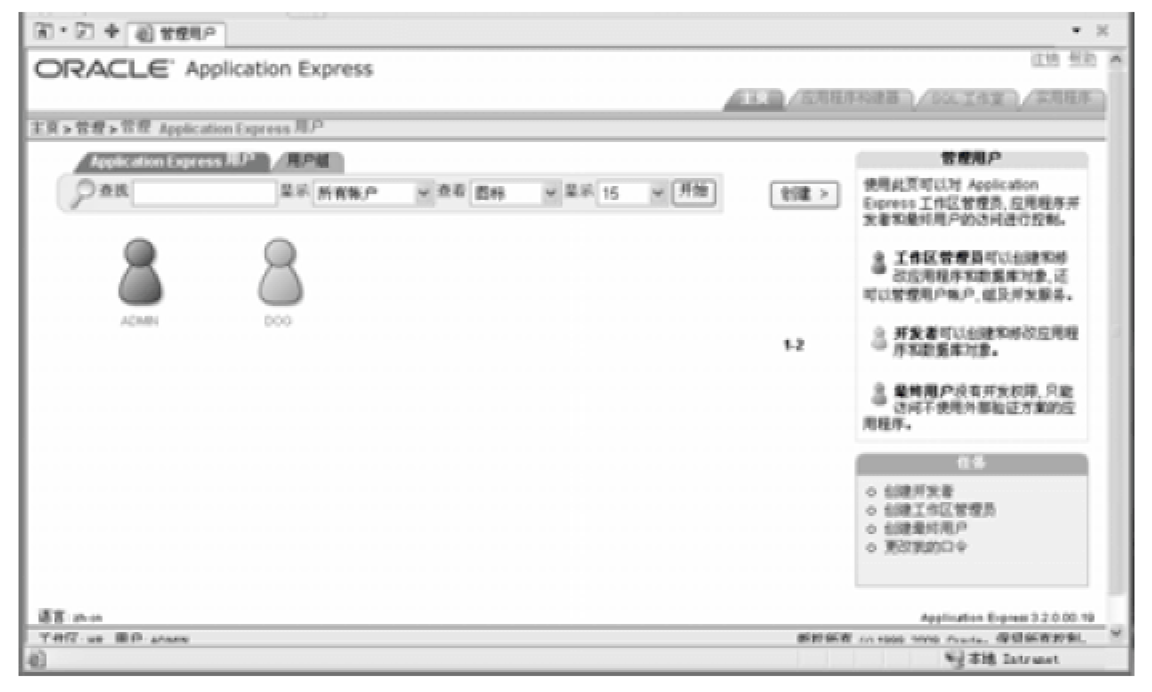


图 19.5

- 监视活动：单击该超链接可以用来监督页面信息和应用程序的变化，如图 19.6 所示。

（2）更改口令

单击“更改口令”超链接将打开一个弹出式窗口，如图 19.7 所示，在这个窗口中可以修改口令。



图　19.6

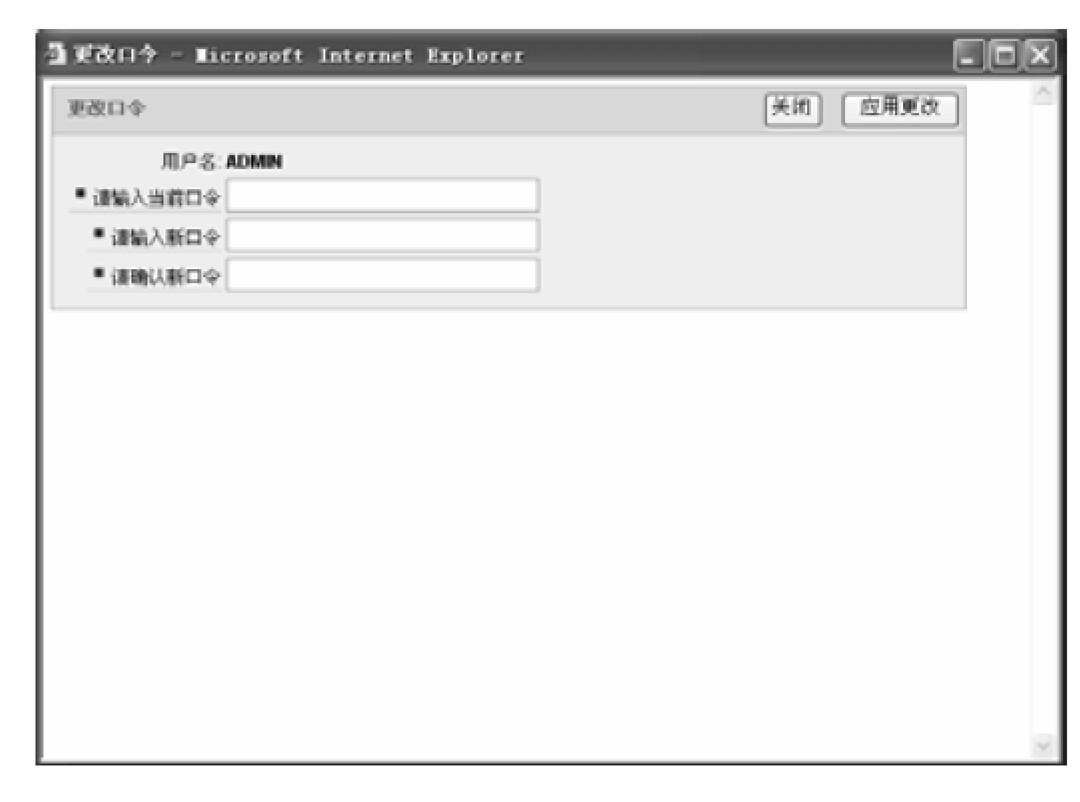


图　19.7

（3）关于 Application Express

单击“关于 Application Express”超链接将看到 Oracle Application Express 和数据库的版本和配置信息，如图 19.8 所示。

2．“移植”列表

“移植”列表出现在工作区主页右侧“管理列表”下，使用该列表中的“应用程序移植”超链接可以将一个微软的 Access 应用系统迁移成 Oracle Application Express。单击这个超链接将看到如图 19.9 所示的应用程序移植网页。



图　19.8

图　19.9

3．“工作区方案”列表

“工作区方案”列表出现在工作区主页右侧“移植”列表下，它显示了与这个工作区相关并可以访问的数据库模式，如图 19.10 所示。

4．“链接”列表

“链接”列表包含了可以扩展您 Oracle Application Express 知识资源的一些链接，它出现在工作区主页右侧“工作区方案”列表下，如图 19.10 所示。读者在使用这些超链接时要先将您的计算机连接到互联网上。



图　19.10

- Oracle 技术网：单击该超链接将打开 Oracle 技术网的 Oracle Application Express 章节。使用这一网页可以访问额外的 Oracle Application Express 信息和资源，如图 19.11 所示。
- 论坛：单击该超链接将打开 Oracle Application Express 论坛，如图 19.12 所示。如果想搜寻问题的答案或回答其他用户的问题，该论坛是一个很棒的资源。



图 19.11

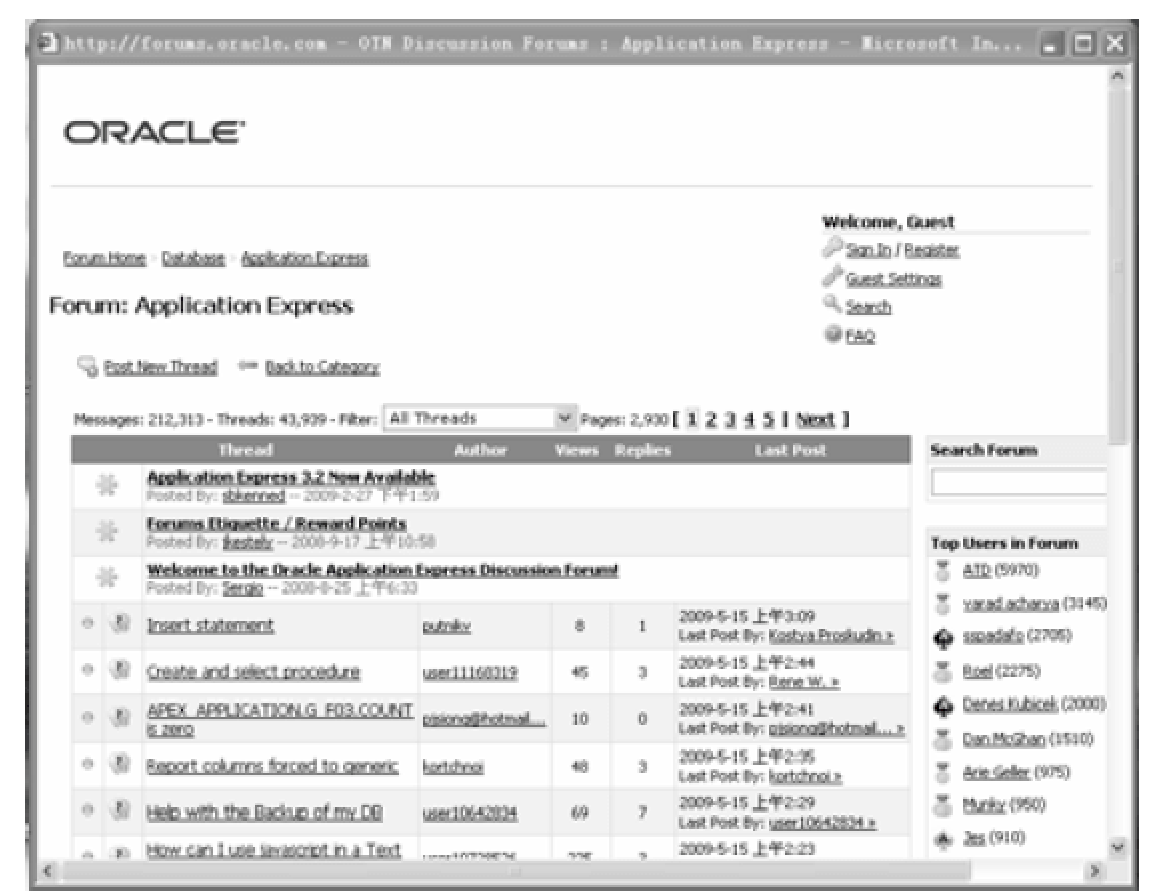


图 19.12

- 用户指南：单击该超链接将打开一个基于 HTML 的帮助系统，如图 19.13 所示。也可以通过单击 Oracle Application Express 任何页面右上角的“帮助”超链接来访问帮助，如图 19.14 所示。

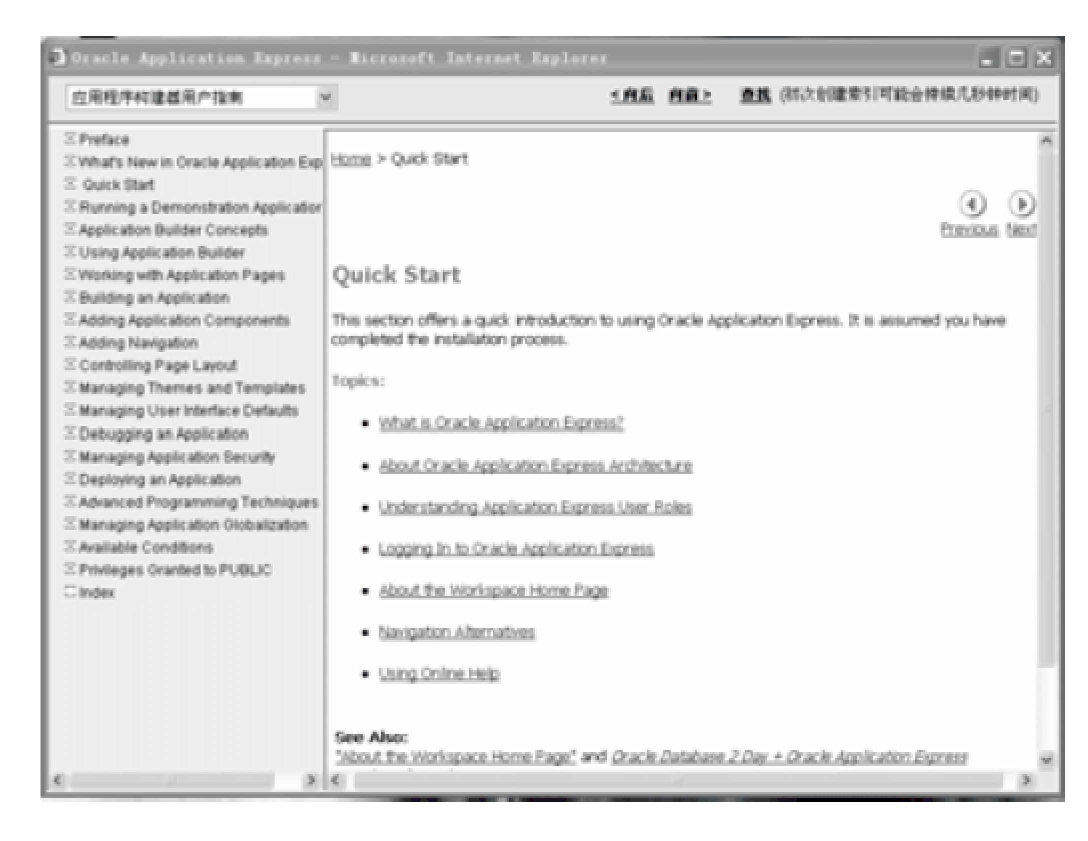


图 19.13

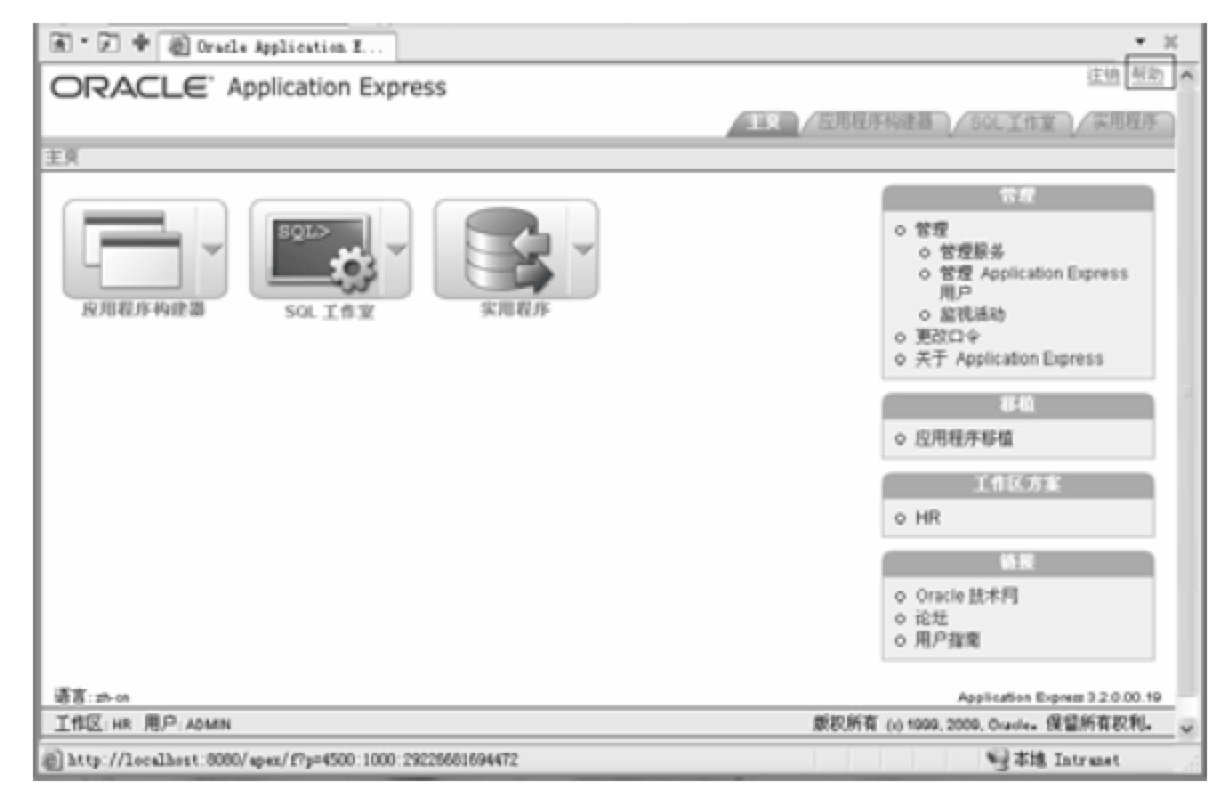


图 19.14

19.2 使用SQL工作室与数据库交互

SQL 工作室是一个访问数据库的工具，它使用户能够通过网络浏览器直接与数据库进行交互，如查看和管理数据库对象。通过使用 SQL 工作室可以轻松地完成如图 19.15 所示的工作。

SQL 工作室提供了一种在应用系统开发期间方便地查询和管理数据库对象的方法，单击 SQL 工作室图标就可以访问它所包含的 4 个工具，如图 19.16 所示。下面分别进行介绍。

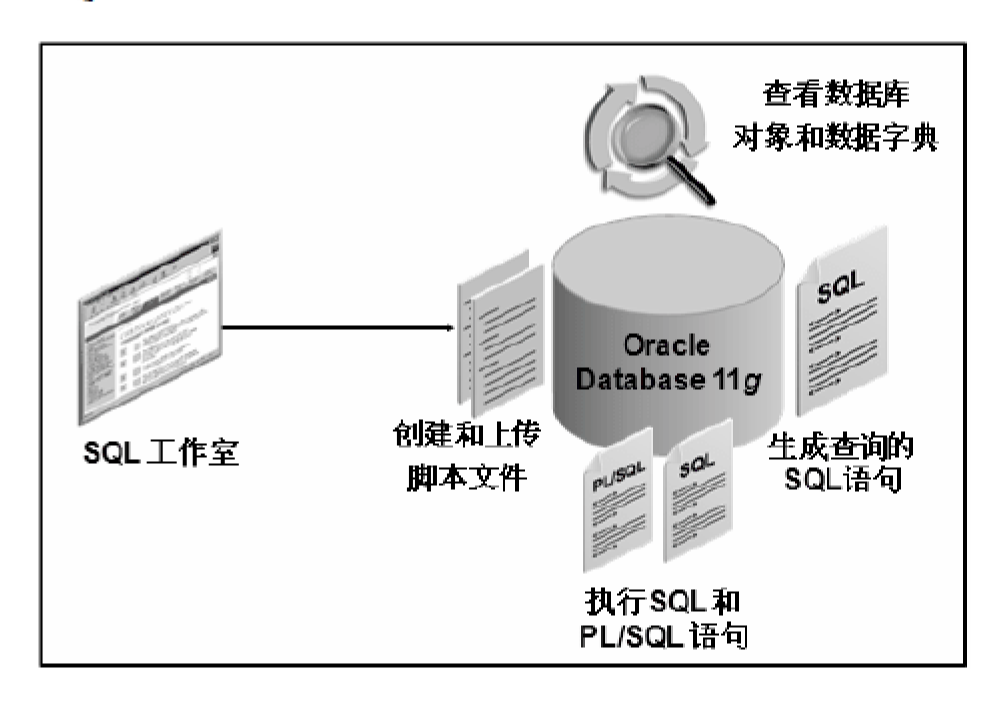


图　19.15

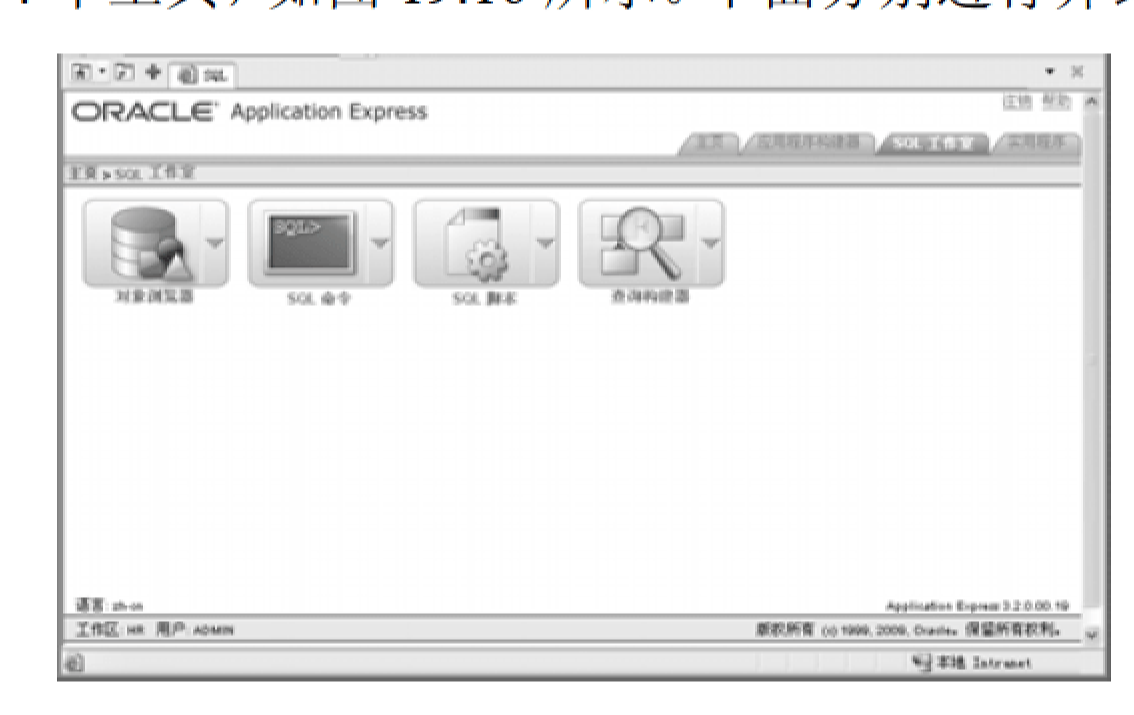


图　19.16

（1）对象浏览器

对象浏览器用于查看、创建、修改、浏览和删除数据库对象，单击“对象浏览器”图标即可打开它，如图 19.17 所示。

选择“创建”选项就可以打开如图 19.18 所示的画面，然后，用户就可以选择任何数据库对象的图标，在向导的引导下轻松地创建对象。

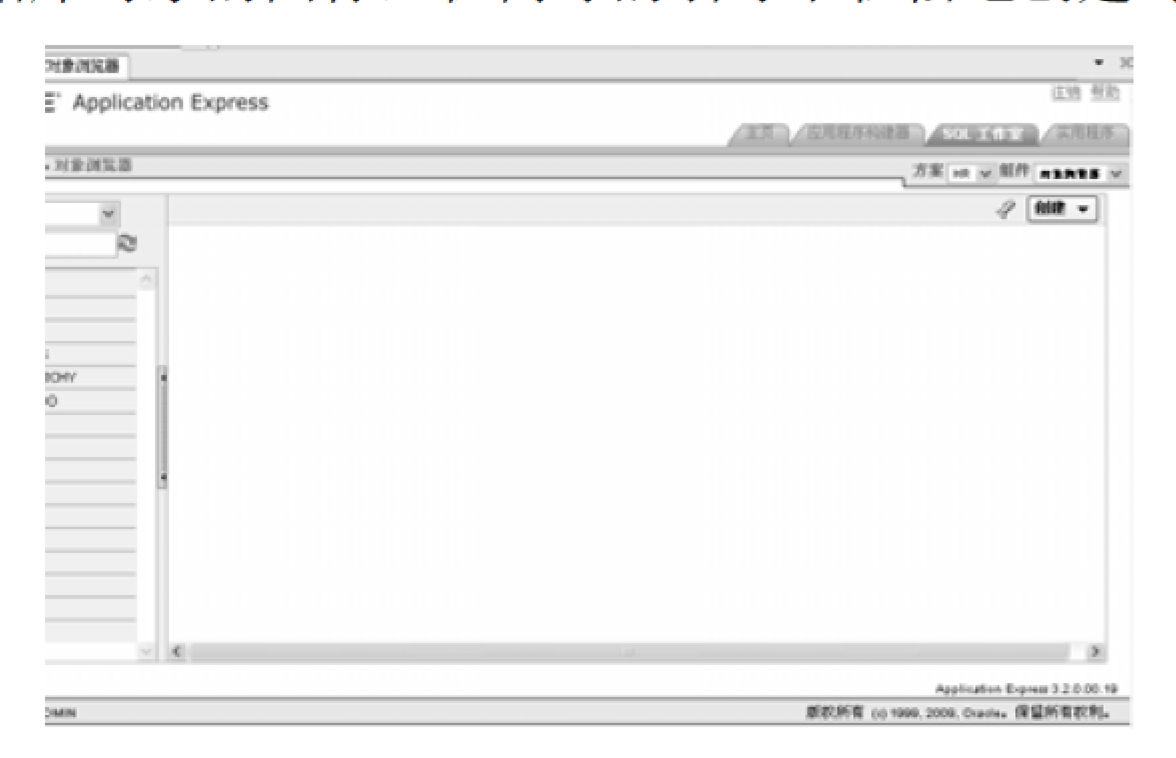


图　19.17



图　19.18

也可以通过选择某个表（如 EMP）来查看这个表的信息，然后可以选择“表”选项卡查看该表的结构，如图 19.19 所示。

如果选择“约束条件”选项卡，就可以获得基于该表的所有约束的信息，如图 19.20 所示。



图　19.19



图　19.20

如果选择 SQL 选项卡，就可以获得创建这个表所需的 DDL 语句，如图 19.21 所示，您也可以将这些 DDL 语句通过复制、粘贴的方法存储到文件中。利用这一简单易学的方法您就可以轻而易举地导出其他人的设计，也就可以踩在别人的肩膀上，爬得更高、升得更快。

以上我们仅仅介绍了对象浏览器的几个常用的功能，其实利用这个工具可以完成更多的工作、浏览到更多的信息，感兴趣的读者可以通过实践很快地掌握。

（2）SQL 命令处理器

在 SQL 命令处理器中可以运行 SQL 语句和匿名的 PL/SQL 语句，也可以运行脚本文件和存储查询语句。单击“SQL 命令处理器”图标将打开如图 19.22 所示的页面。

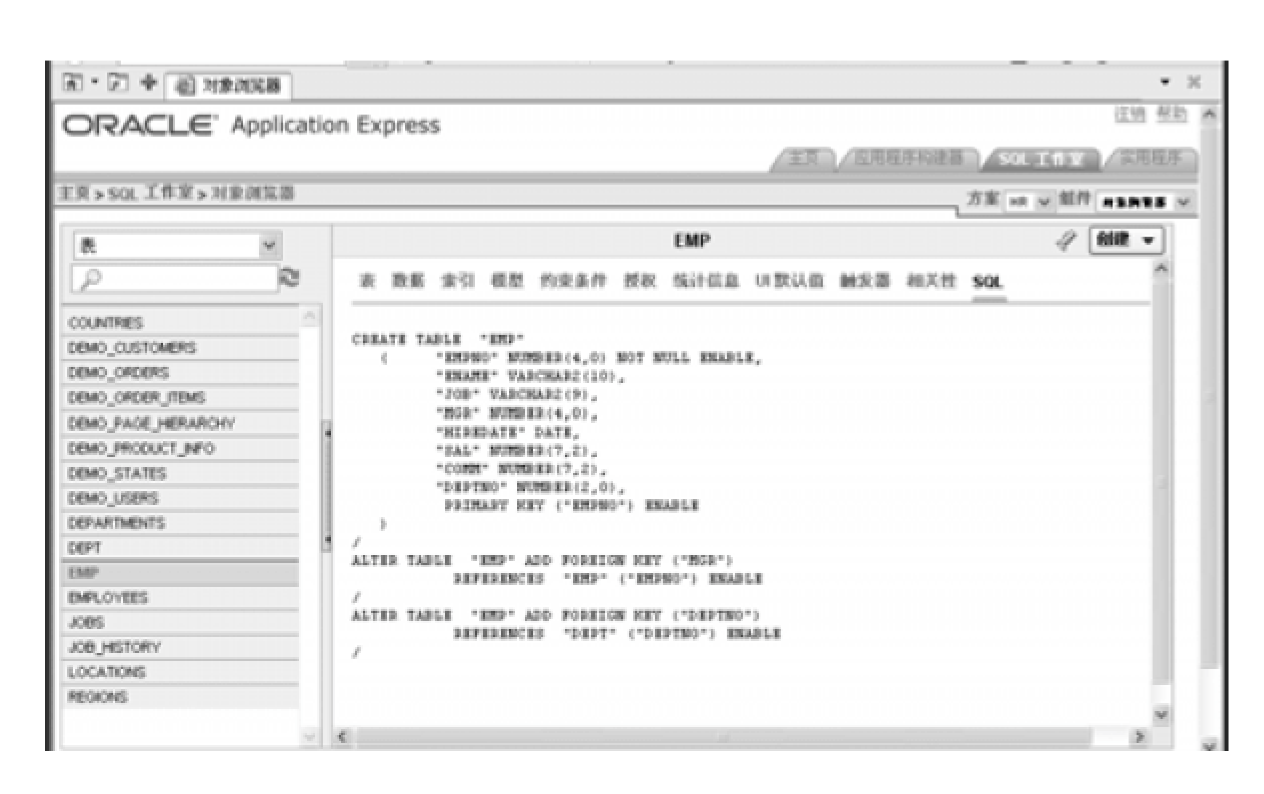


图 19.21



图 19.22

在命令窗口输入例 19-1 的 SQL 查询语句后，单击“运行”按钮就会很快地得到查询的结果，如图 19.23 所示。

例 19-1

```
select ename, job, sal, dname
from emp, dept
where emp.deptno = dept.deptno
```

如果想知道以前执行过的 SQL 语句，可以选择“历史记录”选项卡，然后就可以获得过去执行过的 SQL 语句，如图 19.24 所示。



图 19.23

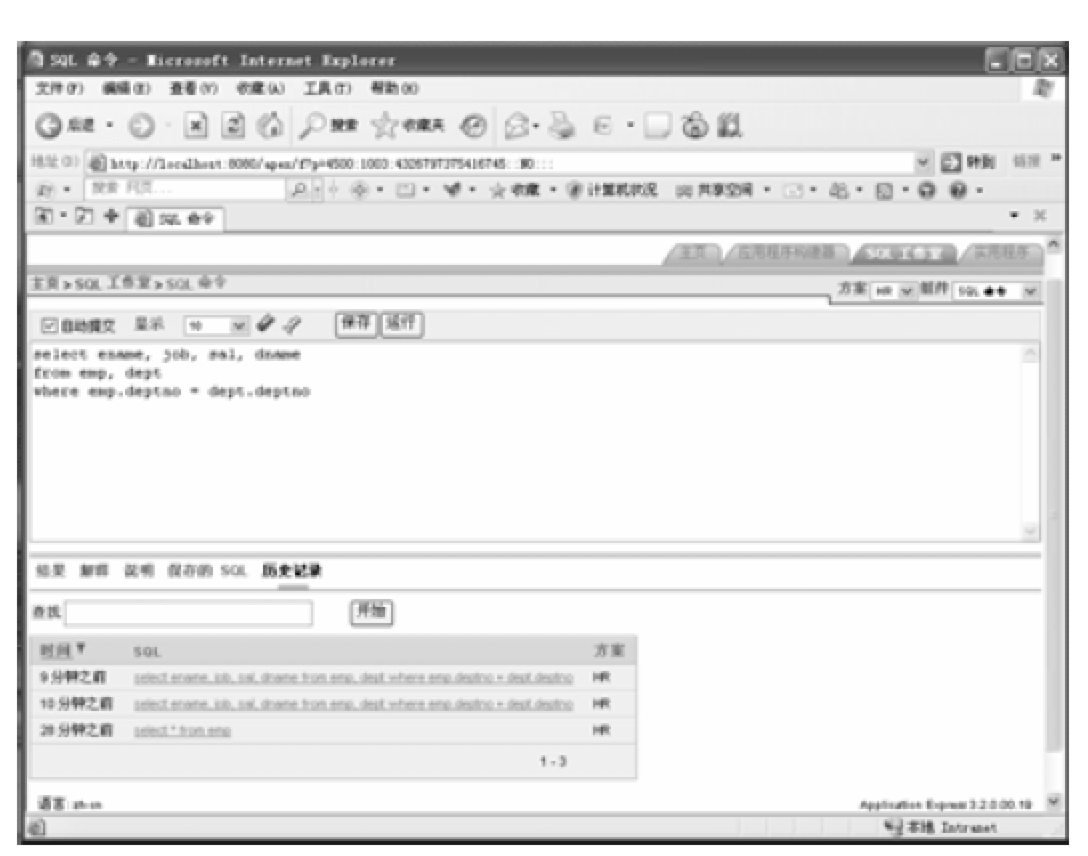


图 19.24

如果想知道 SQL 语句的执行计划，选择“解释”选项卡，就可以获得 SQL 语句的执行计划，如图 19.25 所示。

利用所获得的 SQL 语句的执行计划就可以分析 SQL 语句执行的具体步骤，最终优化 SQL 语句。使用 Oracle Application Express 是不是使工作轻松多了？这也正是“手巧不如家伙妙”。

（3）SQL 脚本

可以用 SQL 脚本来创建、编辑、查看、运行和删除脚本文件，也可以将脚本上传到本地文件系统上，或从本地文件系统下载脚本。单击“SQL 脚本”图标将打开如图 19.26 所示的页面。



图　19.25



图　19.26

（4）查询构造器

利用查询构造器这一图形化的用户界面，用户可以在只有很少甚至没有 SQL 知识的情况下，创建复杂的查询语句。使用这一工具可以轻松地搜寻和过滤数据库对象、选择对象和列、创建对象之间的关系、查看格式化的查询结果和储存查询。单击“查询构造器”图标将打开如图 19.27 所示的页面。

为了演示如何使用查询构造器来方便地构造查询语句，选择 EMP 表并选择 ENAME、JOB、SAL 和 DEPTNO 4 列。然后选择 DEPT 表并选择 DNAME 和 LOC 两列。选择“条件”选项卡，并在 EMP 对象的 DEPTNO 列所对应的条件列中输入“=dept.deptno”，如图 19.28 所示。

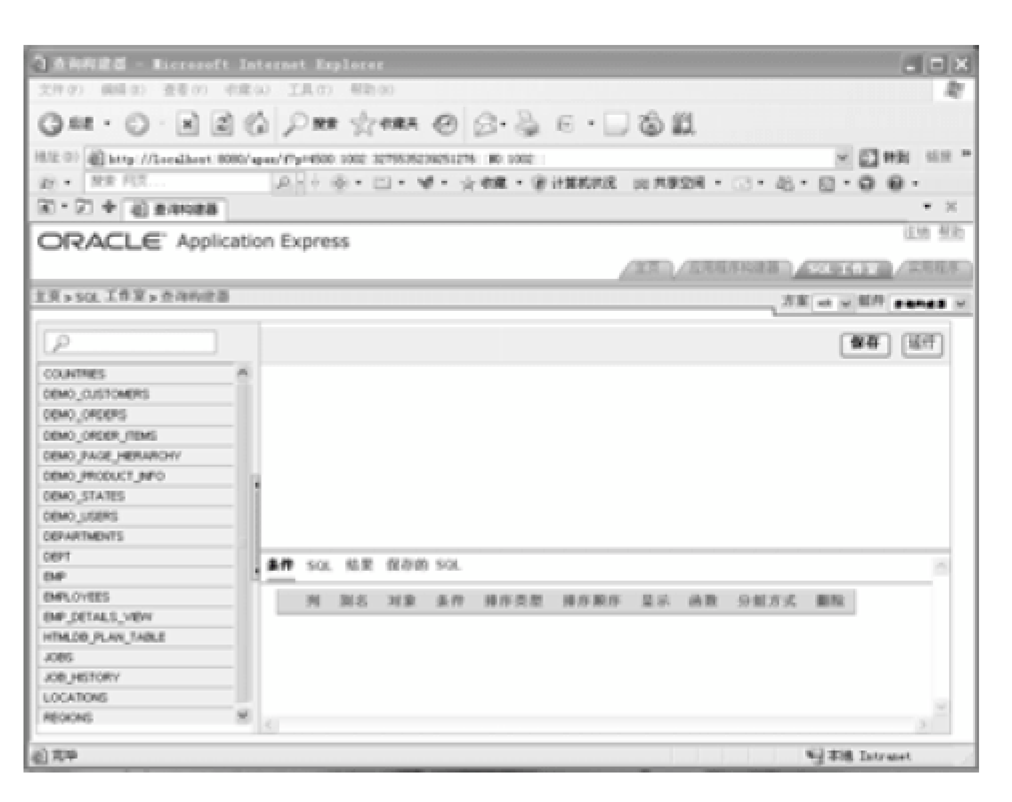


图　19.27

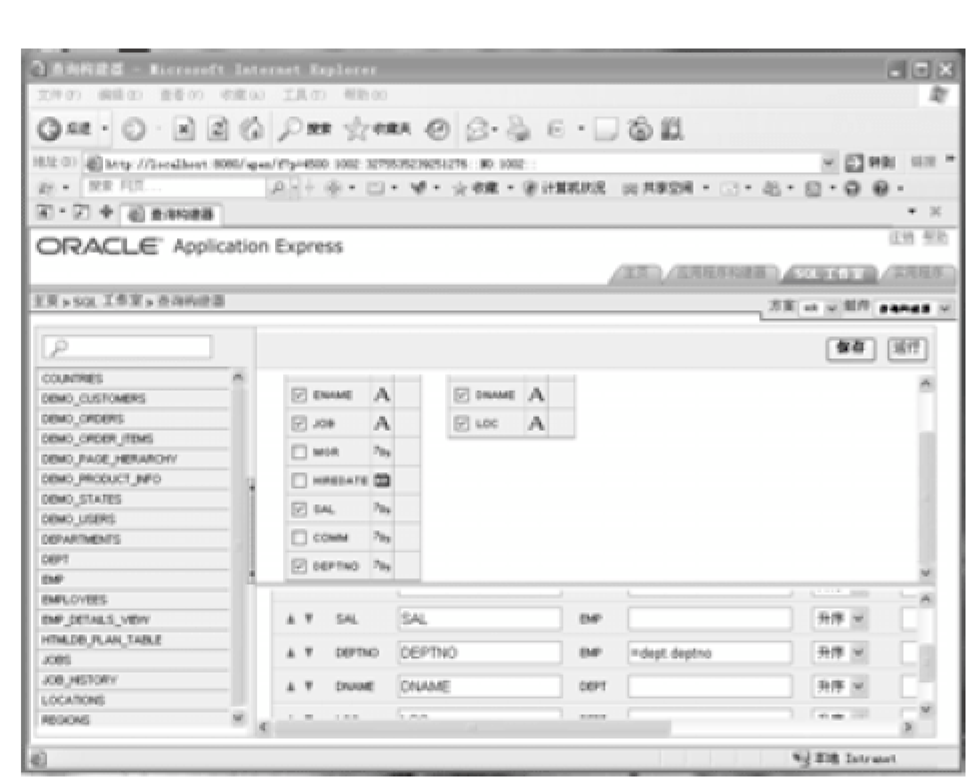


图　19.28

单击“运行”按钮就会生成查询语句并运行该语句，如图 19.29 所示。

如果您选择 SQL 选项卡就会获得查询的 SQL 语句，如图 19.30 所示。现在读者应该相信即使没有 SQL 的知识也可以访问数据库中的数据，因为使用查询构造器可以通过简单的鼠标点击和拖拉及少量、简单的输入就可以生成复杂的 SQL 语句。

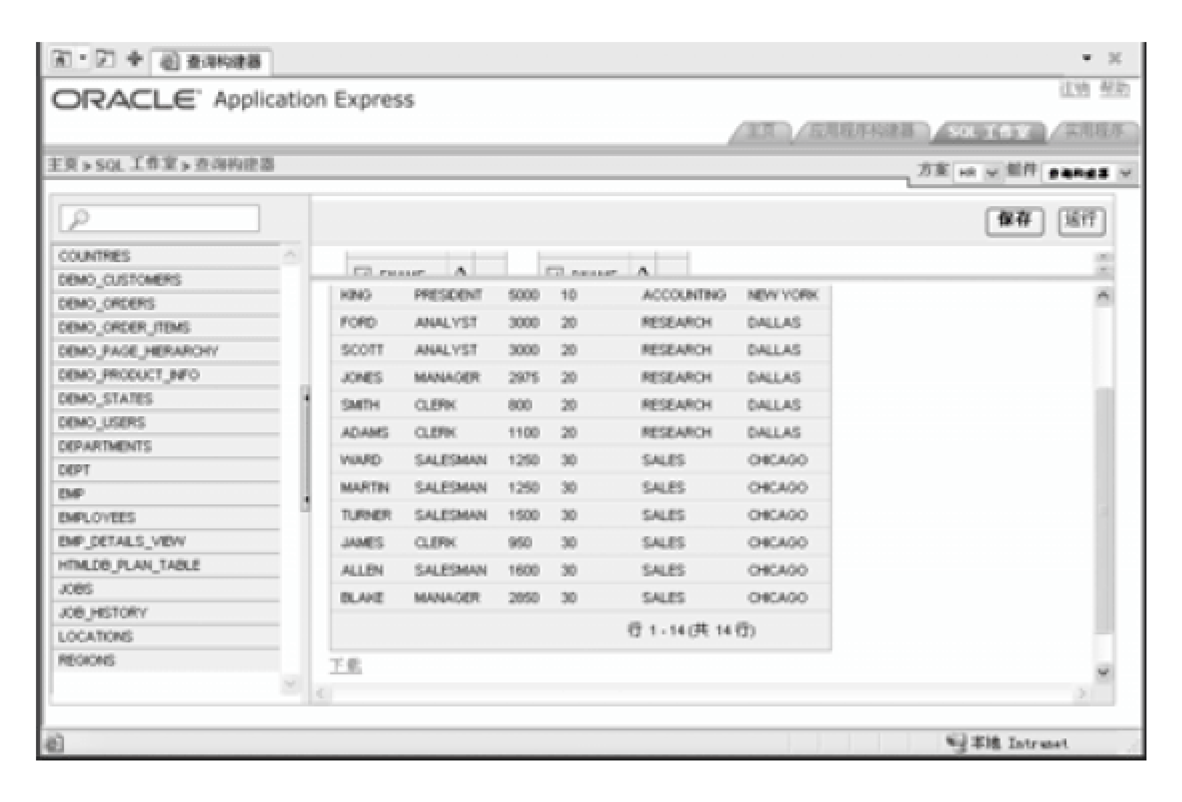


图 19.29

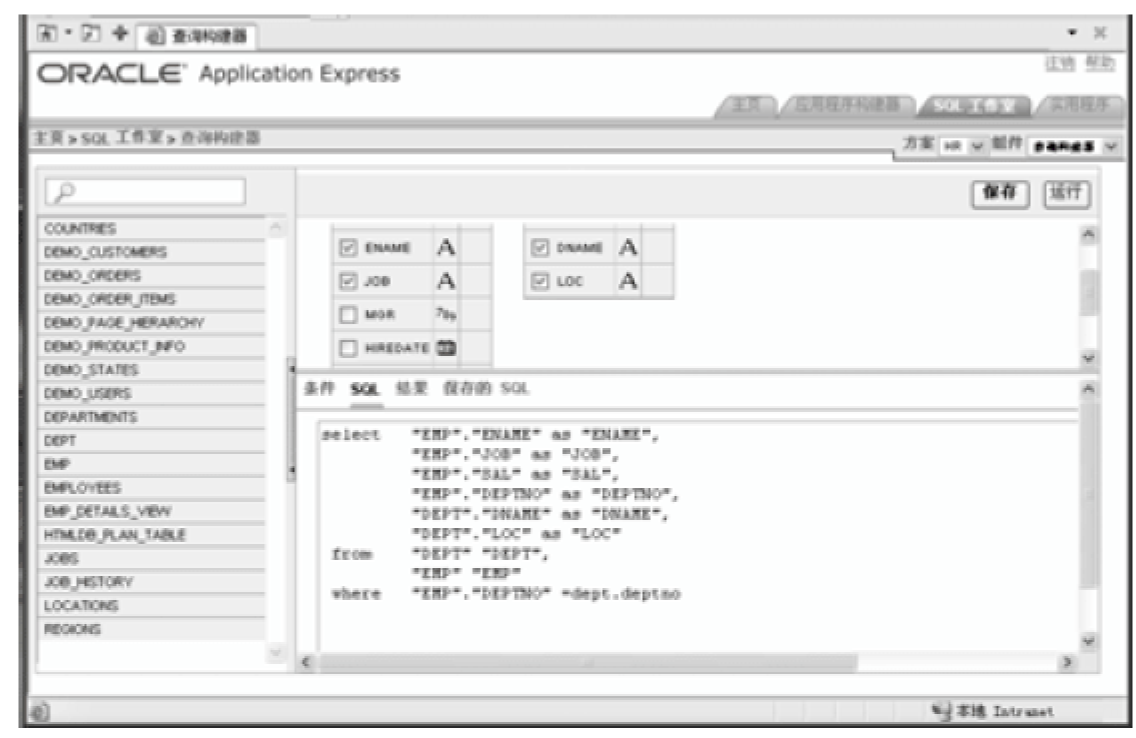


图 19.30

读者现在就可以使用复制和粘贴的方法将如图 19.30 中的 SQL 语句存储到一个 SQL 脚本文件中。原来使用工具进行系统开发就这么容易。尽管在 SQL 部分的学习中我们一直使用命令，但是如果读者将来从事的是开发工作，使用图形化的工具将大大地提高开发工作的效率和准确性。

其实，除了 SQL 语言，在数据库领域还存在另一种广泛使用的语言 QBE（Query By Example），这种语言也被称为图形化查询语言，而 SQL 则被称为命令行查询语言。使用查询构造器访问数据库应该属于 QBE 语言。

19.3 应用程序构建器

应用程序构建器是 Oracle Application Express 中最重要的、也是核心的构件，因为您所设计的任何以数据库为中心的网络应用程序都需要用户接口、应用程序部件和应用程序逻辑，而所有这一切都要使用应用程序构建器来开发。图 19.31 是开发人员利用应用程序构建器来开发以数据库为中心的互联网应用程序的示意图。

应用程序构建器以最优的方式将 HTML 用户接口组装在数据库对象（如表、视图和存储过程）之上。应用程序构建器提供了创建互联网应用程序所需的所有组件，它也加快了应用程序的开发，使用户能够将精力集中在他们的业务（功能）上，而不是应用程序实现的细节上。

每一个 Oracle Application Express 应用程序实际上就是一些利用选项卡、超链接或按钮连接起来的网页的一个集合。

要使用应用程序构建器的组件开始创建以数据库为中心的互联网应用程序，首先要登录 Oracle Application Express 并在首页中单击应用程序构建器的图标，如图 19.32 所示。

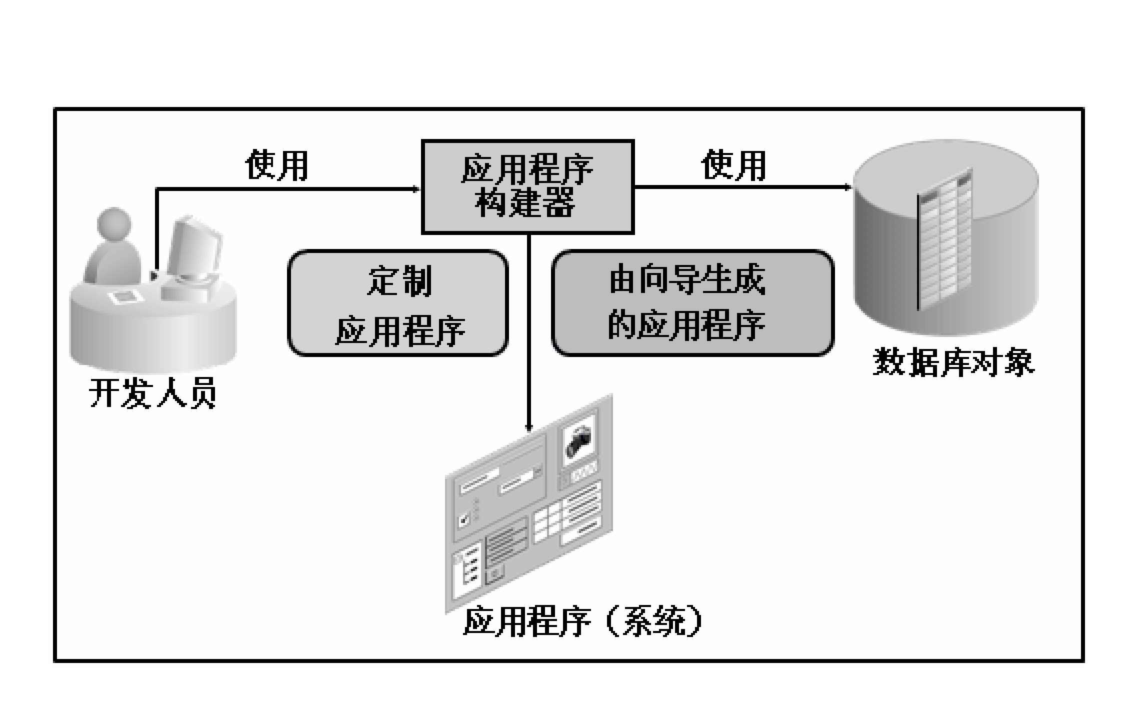


图　19.31



图　19.32

之后就会进入应用程序构建器网页，在这个网页中包括了在该工作区中的所有应用程序。由于我们之前并未开发任何应用程序，所以在该网页中只有一个 Oracle Application Express 自带的样本应用程序，如图 19.33 所示。

单击这个唯一的应用程序就会进入该应用程序的开发页面，如图 19.34 所示。在该页中包含了这个应用程序的所有网页。



图　19.33



图　19.34

选择任意一个页（我们选择是第 1 行的第 2 个）就会进入该页的定义界面，如图 19.35 所示。

页是一个应用程序的基本构件，其中包含了选项卡、列表、按钮、项和区域等用户接口元素。利用页定义页面，可以查看属于您的应用程序的每一页的定义。页定义页面包括如下 3 个主要的区域（从左到右）。

- 页呈现（区域）：是利用数据库中的数据产生网页的处理过程。当一页被呈现时，它将列出用户接口控制和执行程序的逻辑。
- 页处理（区域）：当页被处理时，列出被评估和执行的逻辑控制（如计算和处理）。
- 共享组件（区域）：列出当前页所使用的组件，该页在您的应用程序中也可以被其他应用程序所引用。

图 19.36 是一个应用程序与页及用户接口元素（如项、按钮和区域）之间关系的示意图。

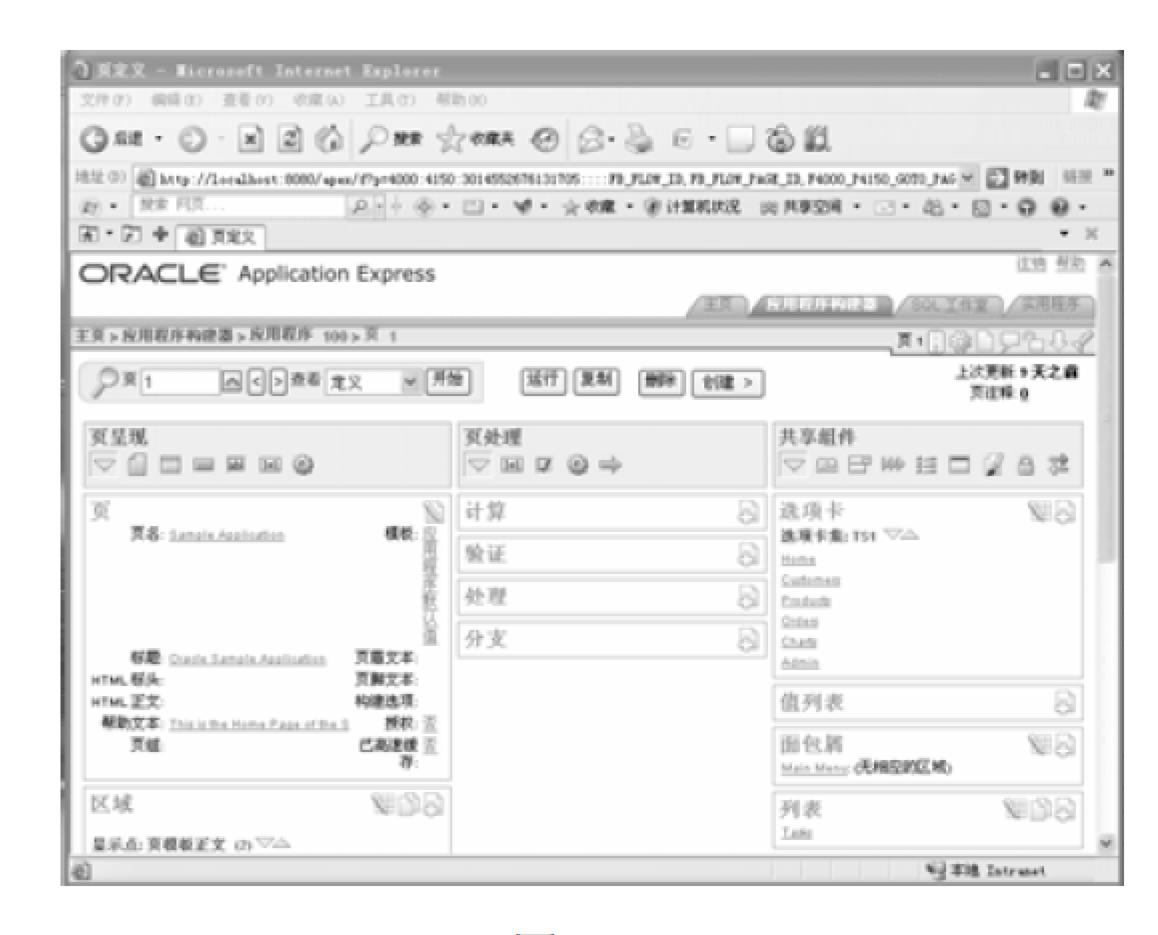


图 19.35

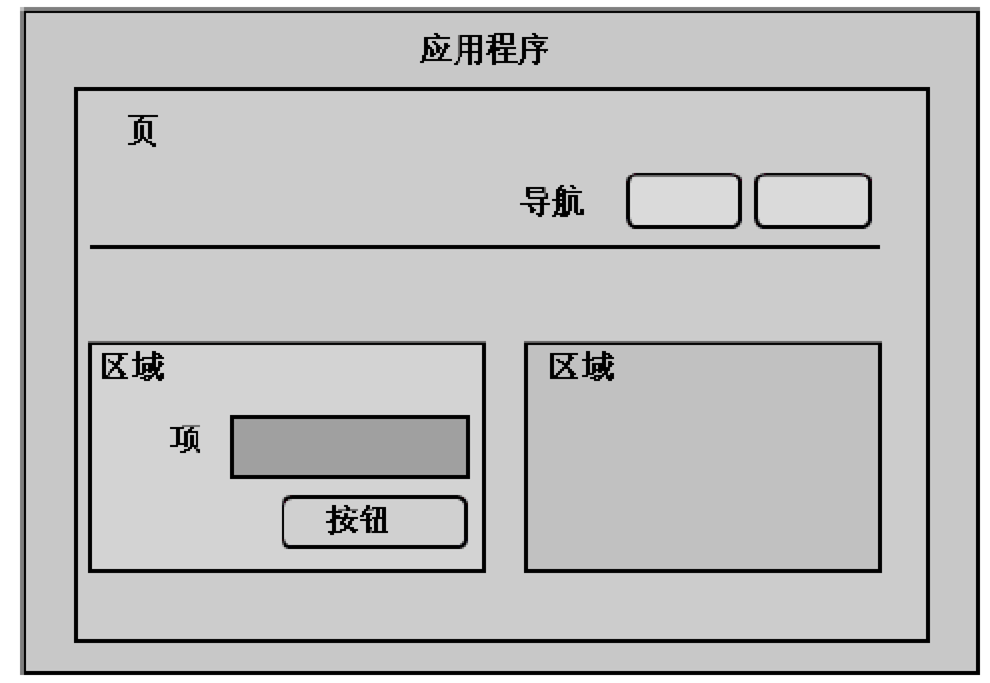


图 19.36

由图 19.36 可以看出，应用程序是由利用导航控制（如选项卡、按钮或链接）连接在一起的一组数据库驱动的网页所组成的。而页是应用程序的最基本的构件，页又可以被进一步分成若干个区域。

每一个区域是页的一部分，其中区域中包含了显示的内容。区域所显示的内容是由区域的信息来源所决定的。例如，一个区域既可以包含一个基于 SQL 查询的报表，也可以包含静态 HTML。区域可以包含如下的元素。

- 项：可以是正文字段、正文区域、组合框和复选框等。
- 按钮：将直接转到某一指定的页或网络地址，也可以发送和处理信息。
- 菜单：提供了分层的导航。

导航选项卡放在区域的外面可以使用户能够在一个应用程序的不同页之间切换。为了使读者更容易理解它们之间的关系，我们回到如图 19.35 所示的页面，并单击“运行”按钮，如图 19.37 所示。

之后就会出现如图 19.38 所示的页面。在 User Name 文本框中输入“demo”（这是 Express 自动安装的，其中有一些演示的例子），在 Password 文本框中输入“hr”（默认为工作区的名字），然后单击 Login 按钮。



图 19.37

图 19.38

之后就会出现如图 19.39 所示的页面，其中，页面右上角的选项卡就是导航选项卡。用户可以通过选择不同的导航选项卡方便地切换到不同的网页。

介绍完了应用程序构建器和 SQL 工作室之后，本来应该介绍实用程序（Utilities）。但是我们将实用程序的介绍放在后面，因为在接下来的章节中还用不到这部分的内容。

在结束本章之前，您可以退回到 Oracle Application Express 的登录页面或重新登录 Express，如图 19.40 所示。

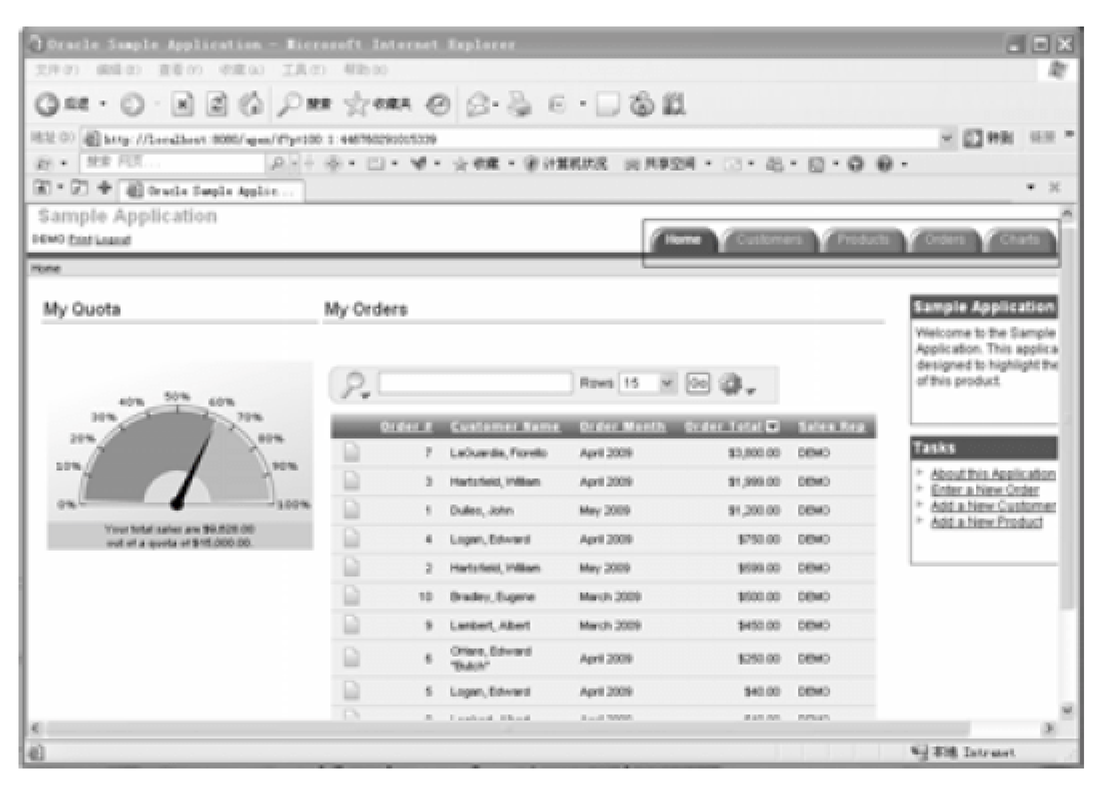


图　19.39



图　19.40

此时，您可以单击“单击此处以了解如何入门”超链接进入如图 19.41 所示的页面。

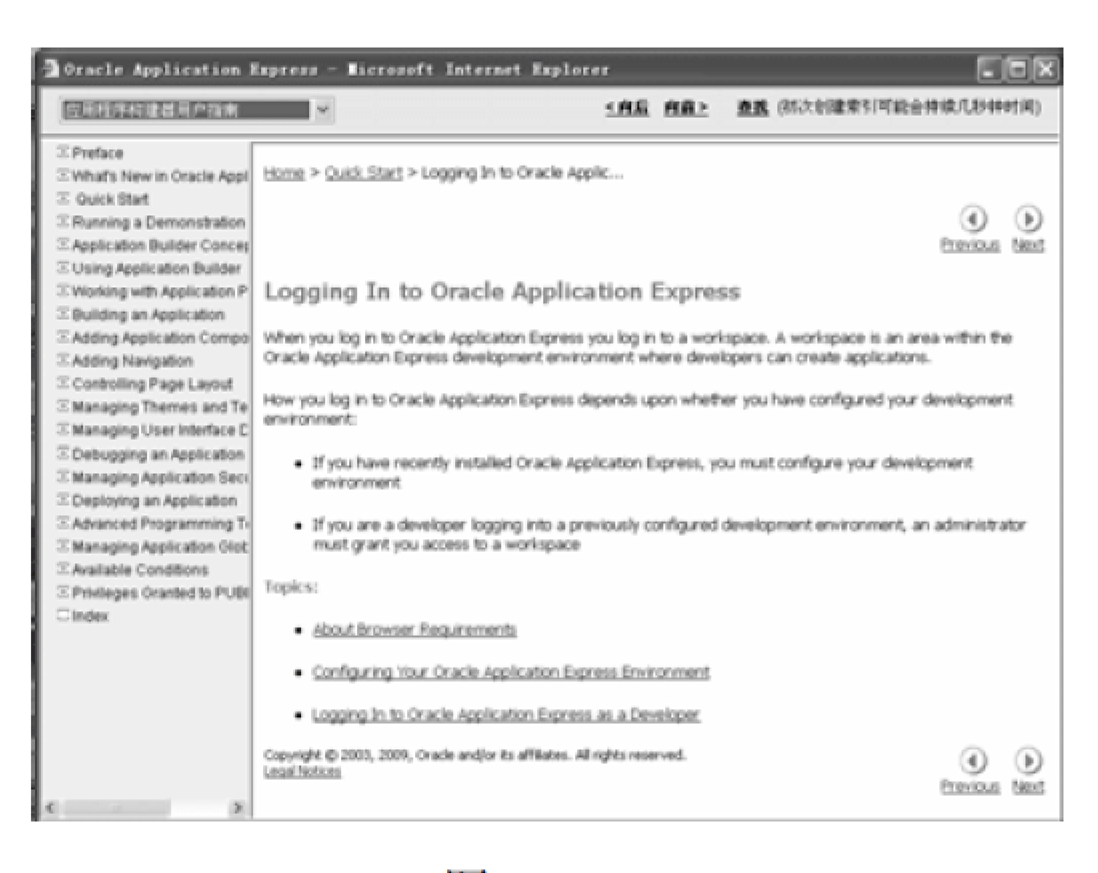


图　19.41

您可以通过这个页面来快速地了解和学习 Oracle Application Express 的使用，不过前提是您需要具有一定的英文水平。接下来您可以试着使用 Oracle 快速 Web 应用开发工具开始创建您的第 1 个基于 Oracle 数据库的网页。

第20章

创建和预览 Express 网页

在本章和接下来的几章中，读者将使用应用程序构建器创建一个人力资源管理的应用程序，其数据来自 Oracle 数据库的 HR 用户。同时，开发（创建）的过程中，您也将在不同的阶段使用它所提供的多种功能不断地修改、调整和预览您所开发的应用程序（网页）。

为了帮助读者更好地理解 Oracle Application Express（Oracle 快速 Web 应用开发工具）在商业公司的实际应用，我们利用一个叫武大郎烧饼总公司的虚拟公司来介绍怎样使用 Express 开发商业应用系统。下面是该公司的简介:

武大郎烧饼总公司是由武大郎和他的贤妻潘金莲的第 38 代嫡传长孙女所创办的民营企业，以餐饮业为主。公司的创办人在公开场合曾多次宣称，她创建公司的初衷就是要把“武大郎烧饼”这个千年品牌和中华老字号传承下去，并发扬光大，让这个深入人心和飘香了千年的烧饼走进寻常百姓家。

尽管自从该企业创立以来餐饮业就屡遭磨难，先是瘦肉精事件，之后是疯牛病，接下来又是禽流感，最近又是猪流感。但是她的企业却一直以燎原之势持续地超速发展，因为该公司的招牌产品、也是镇店之宝的是“武大郎驴肉火烧”，公司对外声称这种驴肉火烧的秘方是他们的先祖之一潘金莲在梦中经仙人指点所得。现在武大郎烧饼总公司的分店和合作伙伴几乎已经遍布全国。

目前，公司正在运筹下一步的战略，就是走向世界。作为全球化的第一步就是将公司数据库中的信息放在互联网上。接下来您就将为这一拥有千年品牌和承载着许多中华文化的知名企业开发和部署基于互联网的应用系统。

20.1　创建最初的Express应用程序

下面将使用应用程序构建器向导创建您的第 1 个 Express 应用程序，该应用程序只有两页，只定义了应用程序的最基本的功能。

公司的决策层考虑到直接将公司的信息放在环球网上有一定的风险，因为如果出现问题可能会直接影响企业的公共形象。于是决定先将公司的信息放在公司的内网上进行试点，等成熟之后再放在公网上。首先，老板叫您以最快的速度将数据库中所有分公司和部门的信息放到公司的内网上。具体的步骤如下：

（1）登录 Oracle Application Express。为了显得专业（也是为了安全），使用 dog 用户登录，该用户的口令也是 xm_Q1ng，图 20.1 为 dog 用户登录的页面。

（2）单击“登录”按钮后就进入 Oracle Application Express 工作区的主页，如图 20.2 所示。

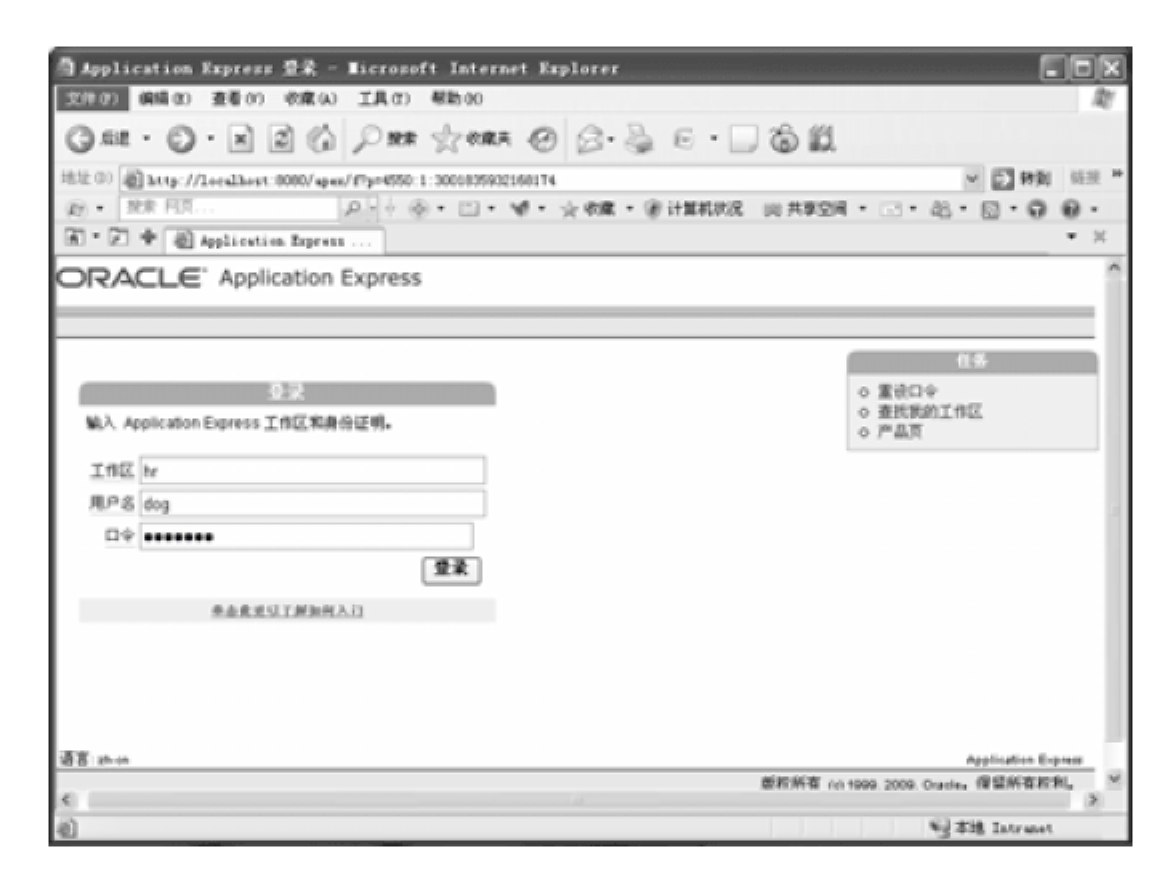


图　20.1

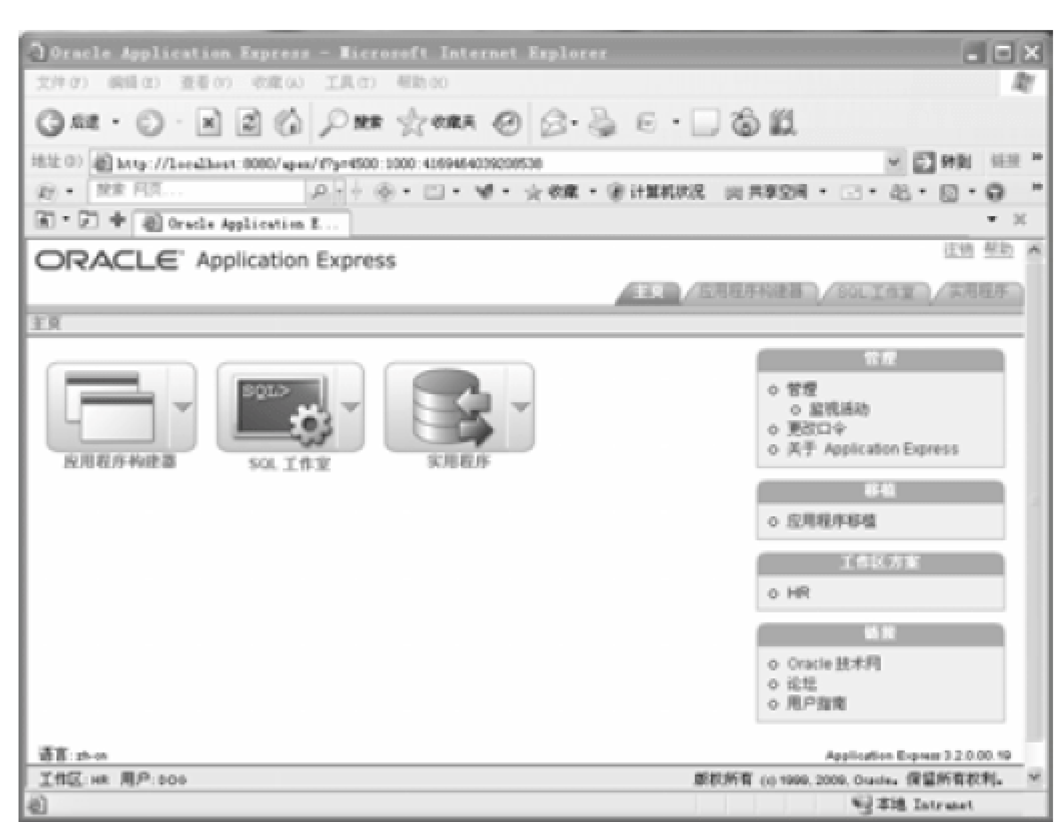


图　20.2

（3）单击“应用程序构建器”图标后，会进入应用程序构建器页面，如图 20.3 所示。

（4）单击“创建”按钮将出现应用程序向导，如图 20.4 所示。



图　20.3



图　20.4

向导中的每一页都要显示向导的标题（图 20.4 中的标题为“创建应用程序”）。每一向导页还要在左边显示一系列信息框，这些信息框代表了执行步骤的顺序，其中加亮部分为正在执行的步骤。通过这些信息开发者可以清楚地知道自己工作的进度。

（5）设置方法。接受默认标题“创建应用程序”后，单击“下一步”按钮，将会进入名称所在的页面，如图 20.5 所示。

（6）设置名称。

① 在“名称”文本框中输入“Jinlian”（您可以使用其他名字）。

② 在“应用程序”文本框中接受默认 ID，系统会自动为您在该工作区中创建的每一个应用程序产生一个唯一的 ID。

③ 在“创建应用程序”栏中接受默认选中的“从头开始”单选按钮。

④ 在“方案”下拉列表框中选择 HR（包含创建网页所需对象的数据库用户）选项。

⑤ 单击“下一步”按钮，将进入页所在的页面，如图 20.6 所示。



图　20.5



图　20.6

（7）设置页，在“添加页”区域进行如下的定义。

① 在“选择页类型”栏中接受默认选中的“空白”单选按钮。

② 在“页名”文本框中输入“主页”。

③ 单击“添加页”按钮。

之后将出现如图 20.7 所示的页面，其主页页面将出现在上部，而且“添加页”区域再次出现，因此您可以继续向最初的应用程序中加入网页。接下来，将添加一个基于 DEPARTMENTS 表的报表。

（8）在“添加页”区域进行如下的定义，如图 20.8 所示。

① 在“选择页类型”栏中选中“报表”单选按钮。

② 在“从属于页”下拉列表框中选择“主页（1）”选项，这一步将在您的应用系统中的网页之间建立起一个层次结构。

③ 在“页源”下拉列表框中选择“表”选项。

④ “表名”处选择 DEPARTMENTS。选择该选项可以显示所有的与该应用系统相关的数据库用户中的表和视图。

⑤ 在“实施”下拉列表框中选择“经典”选项。

⑥ 取消选中“包含分析页”复选框。

⑦ 单击“添加页”按钮。



图　20.7

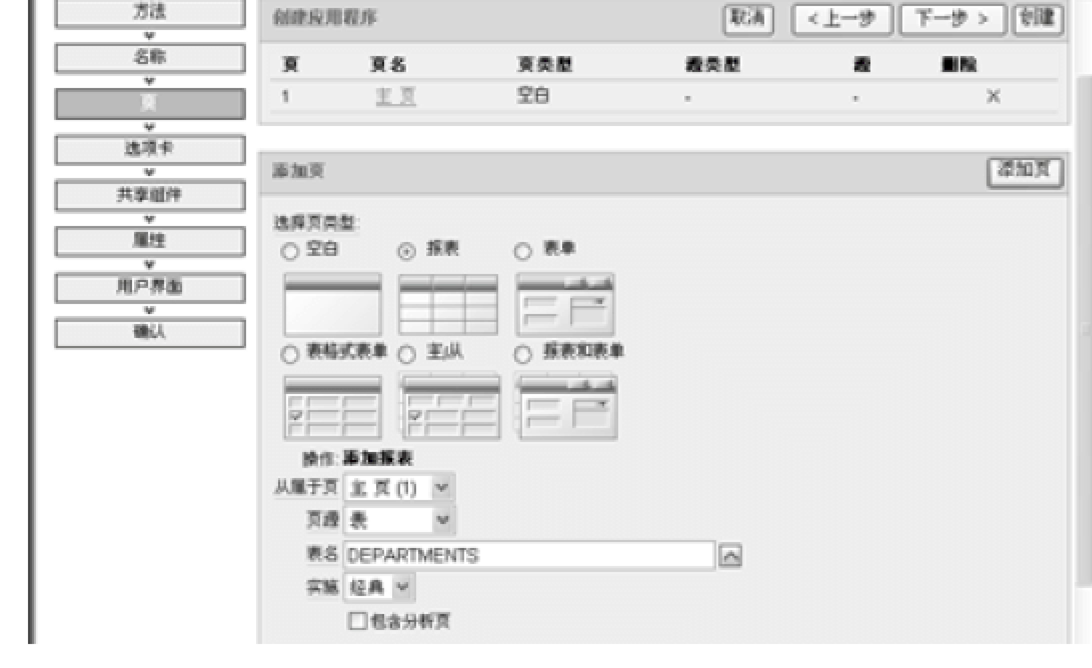


图　20.8

之后，将出现如图 20.9 所示的页面。您在应用系统中创建的所有网页都将按定义的层次列出在该页的上部。接下来，为了使网页更吸引人的眼球，您将默认的网页名 DEPARTMENTS 改为“分公司”。

⑧ 单击 DEPARTMENTS 超链接，将出现如图 20.10 所示的页面。



图　20.9

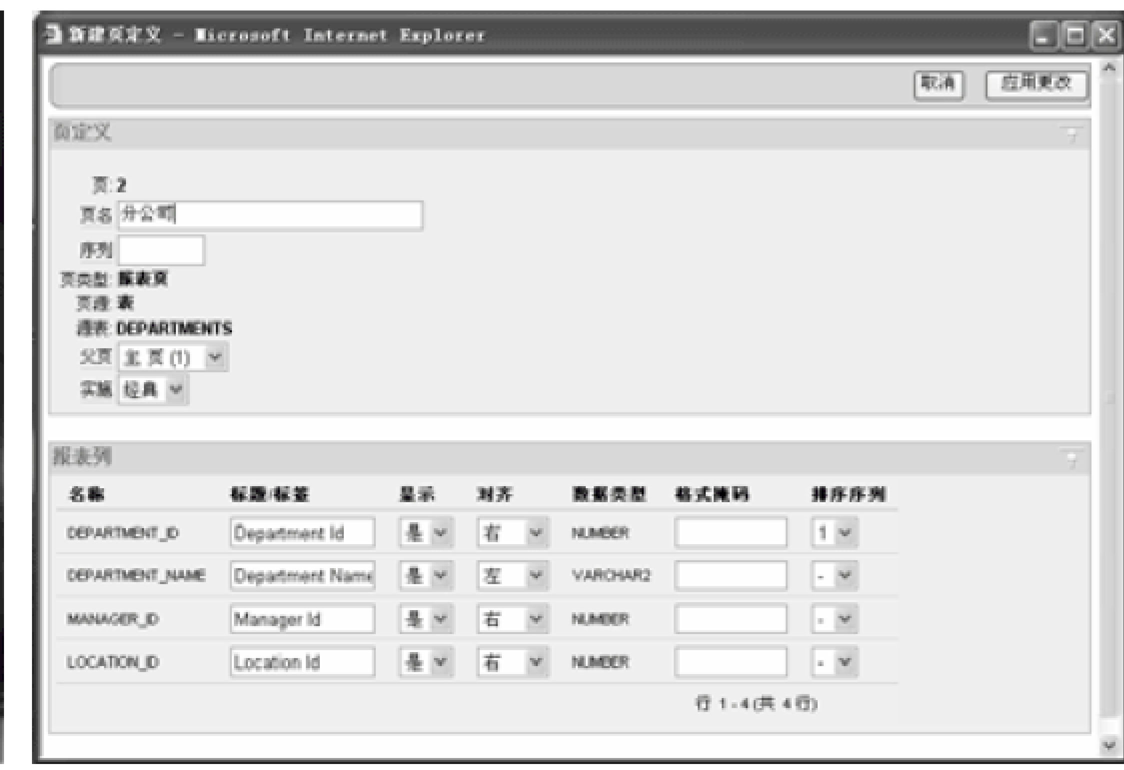


图　20.10

⑨ 在“页定义”区域将“页名”改为“分公司”。

⑩ 单击“应用更改”按钮，将出现如图 20.11 所示的页面。

⑪ 在“创建应用程序”页单击“下一步”按钮，将出现如图 20.12 所示的页面。

到此为止，添加页部分的定义已经全部完成，接着要定义一些应用程序级的设置。



图　20.11

图　20.12

（9）在图 20.12 中的“选项卡”栏中选中“无选项卡”单选按钮，单击“下一步”按钮，将出现如图 20.13 所示的页面。

（10）在“从其他应用程序复制共享组件”栏中接受默认选中的“否”单选按钮并单击“下一步”按钮，将出现如图 20.14 所示的页面。

（11）对这一页中的所有属性都接受默认值并单击“下一步”按钮，之后将出现如图 20.15 所示的页面。

（12）对于用户接口，选择“主题 18”并单击“下一步”按钮，之后将出现如图 20.16 所示的页面。

（13）在图 20.16 中会显示您所做的所有定义，此时，您需要核对一下这些信息。如

果准确无误，就单击“创建”按钮。之后将出现如图 20.17 所示的页面。



图 20.13



图 20.14



图 20.15



图 20.16

现在您已经成功地创建了主页和分公司两个网页，这也是您使用 Oracle Application Express 创建的第 1 个应用程序。在默认情况下它们是以图标方式显示的。

为了显示页的细节而不是图标，可以在“查看”下拉列表框中选择“详细资料”选项并单击“开始”按钮即可，如图 20.18 所示。



图 20.17

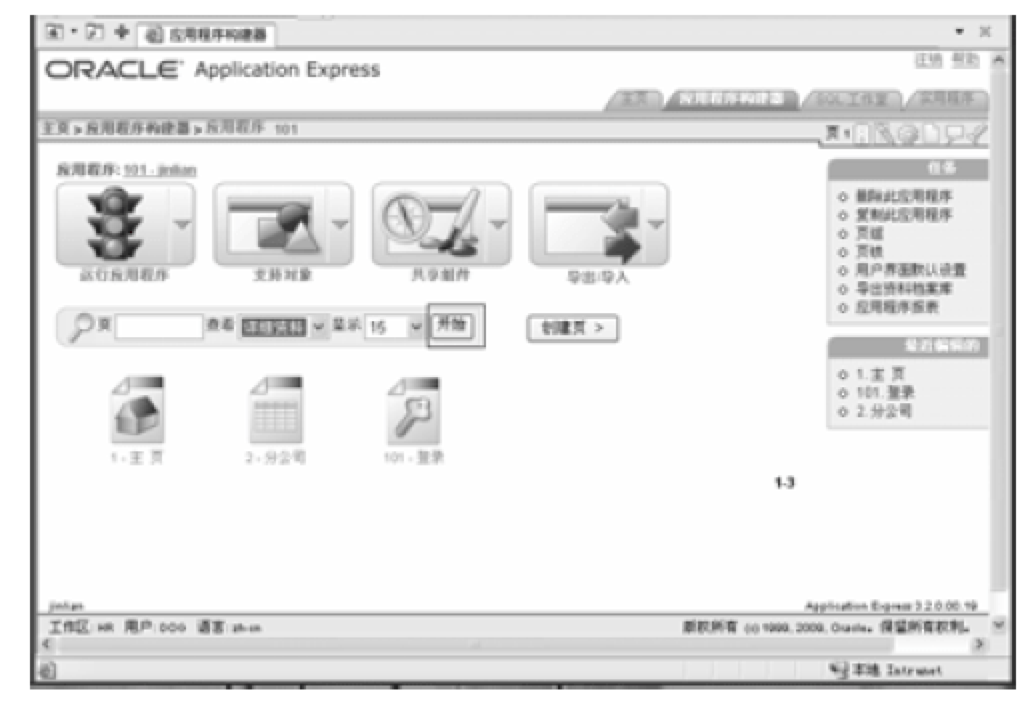


图 20.18

之后就会出现应用程序中所有页面的细节信息，如图 20.19 所示。为了后面的学习方便，现在我们还是切换回默认的图标显示方式。



图　20.19

20.2　预览所建的应用程序

用户可以通过运行所创建的应用程序来预览它。运行应用程序将显示所创建的网页，终端用户将会看到应用程序呈现的版本。当用户创建网页时，他既可以通过运行单独的页、也可以通过运行整个应用程序来运行所创建的这些网页。当用户运行一个网页或应用程序时，Oracle Application Express 引擎将存储在数据库中的数据动态地以可见的 HTML 形式呈现给用户。

选择网页，然后单击“运行页”图标，单击如图 20.20 中右侧以黑框括住的图标即可运行网页。

要运行应用程序，单击“运行应用程序”图标即可，如图 20.21 所示。

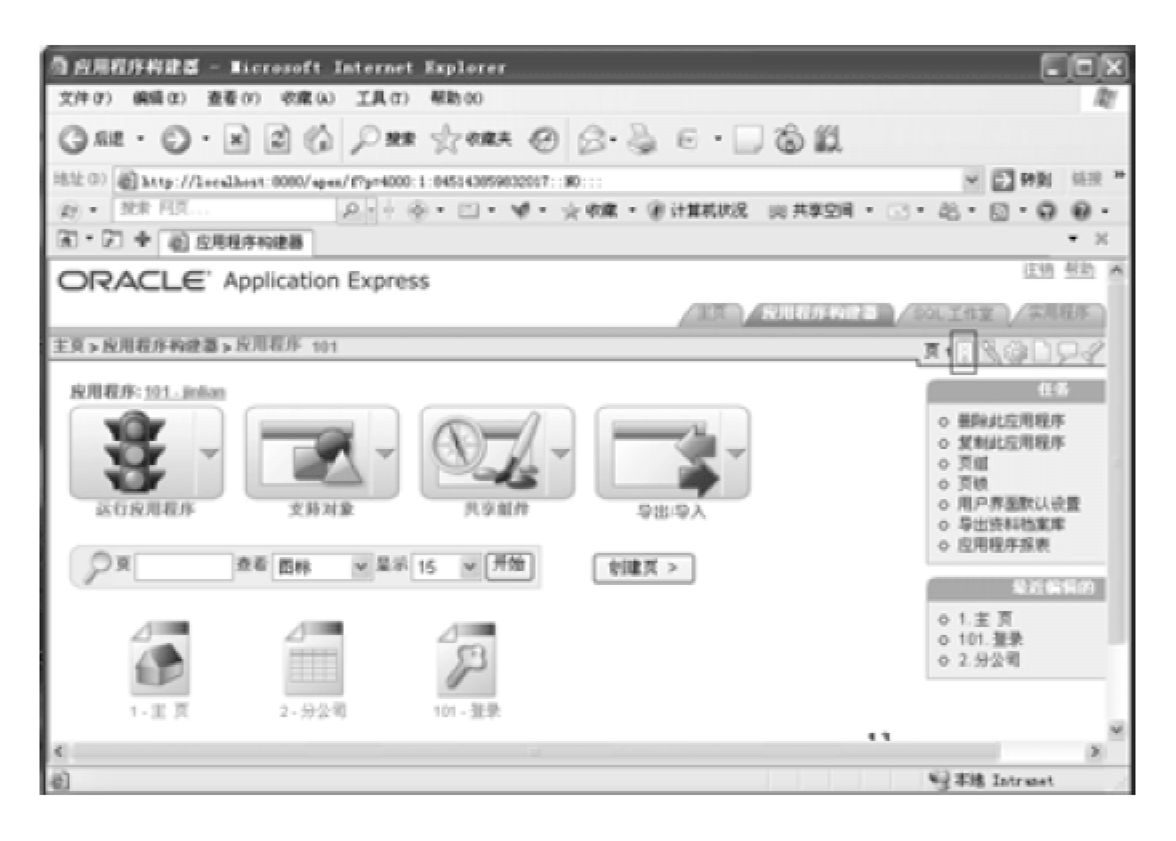






图　20.20　　　　图　20.21

尽管按老板的要求，您已经成功地创建了公司第 1 个互联网应用程序，但是为了慎重起见，在拿给老板审查之前您要先仔细地检查，具体操作步骤如下：

（1）在“应用程序 101”页面，单击“运行应用程序”图标，如图 20.22 所示。

（2）之后就会出现如图 20.23 所示的登录界面。在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”（您可能使用不同的用户名和口令），单击“登录”按钮。

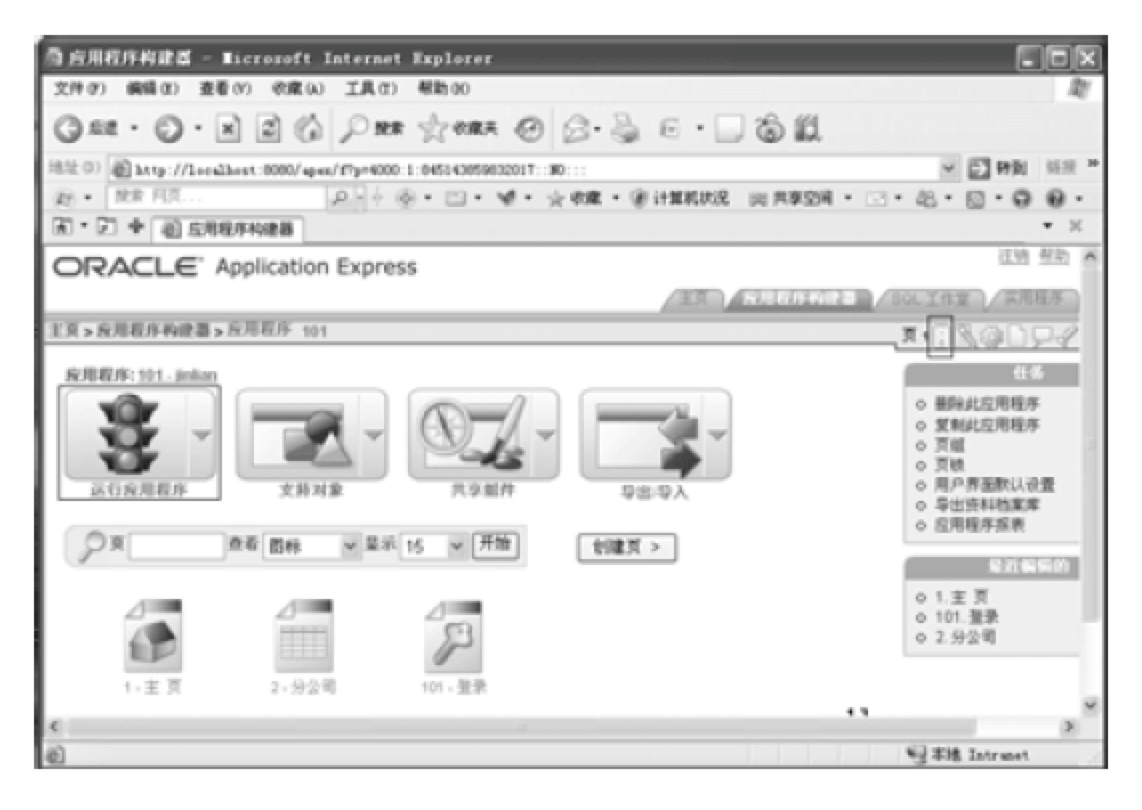


图 20.22

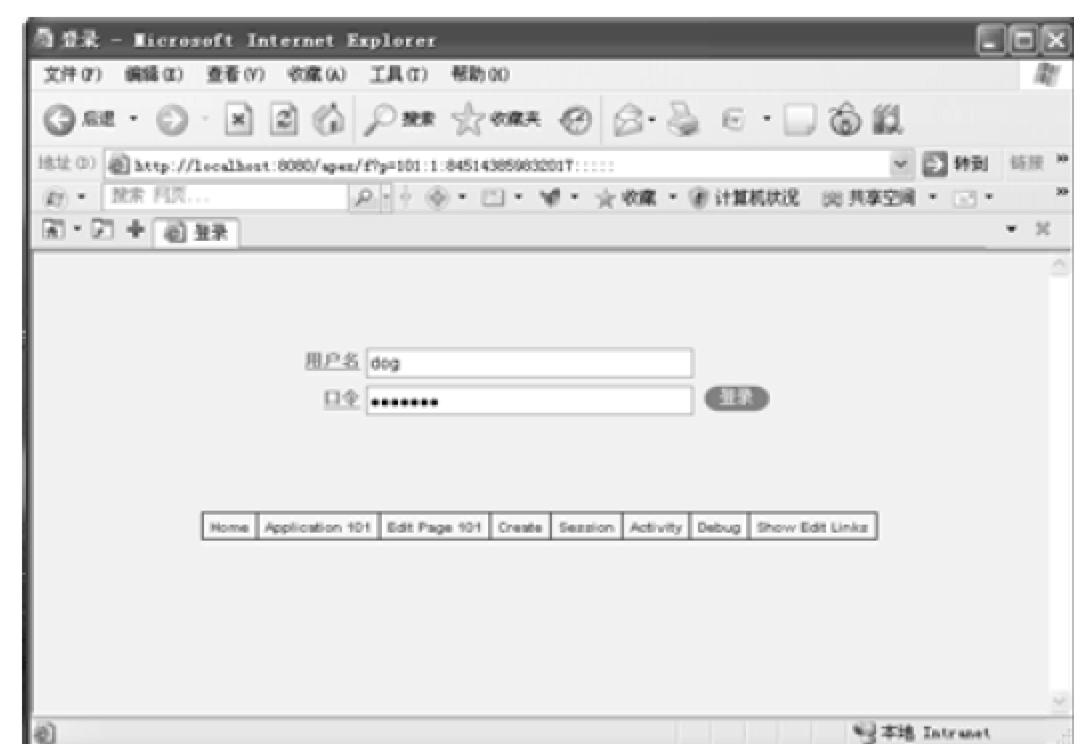


图 20.23

之后就将进入应用程序的主页，如图 20.24 所示。“分公司”超链接出现在主页的导航部分，因为您设置了它在应用程序中的层次。用户名 DOG 出现在该页的左下方。在该页的最下方是开发工具栏。只有当您在开发环境中运行应用程序时才会出现开发工具栏的链接。开发工具栏提供了编辑当前页的快捷方法，可以使用它创建页、区域或页面控制，也可以查看会话的状态，还可以在调试和运行状态之间进行方便的切换。

（3）如果要继续预览应用程序，单击主页上的“分公司”超链接，就可以显示分公司页面，如图 20.25 所示。



图 20.24

Department Id	Department Name	Manager Id	Location Id
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

图 20.25

为了方便用户开发和维护他们的应用程序，Oracle Application Express 提供了一些操作链接和按钮。它们可以使用和快速完成某些特定的工作，下面进行具体介绍。

- 面包屑（以黑框括起来的部分）：显示网页的层次和路径，单击一个面包屑可以移动到之前的页，如图 20.26 所示。
- 排序：单击某一列的列标题就将对数据进行排序，如图 20.27 所示。
- 搜索：输入搜索的关键字可以用来定位表中的记录，搜索是大小写无关的，如图 20.28 所示。例如，在“搜索”文本框输入“Operations”后单击“开始”按钮，就会得到部门名为 Operations 的记录行，如图 20.29 所示。
- 显示：每一页所显示的记录行数，您可以根据需要调整这一数字，如图 20.30 所示。



图　20.26



图　20.27



图　20.28

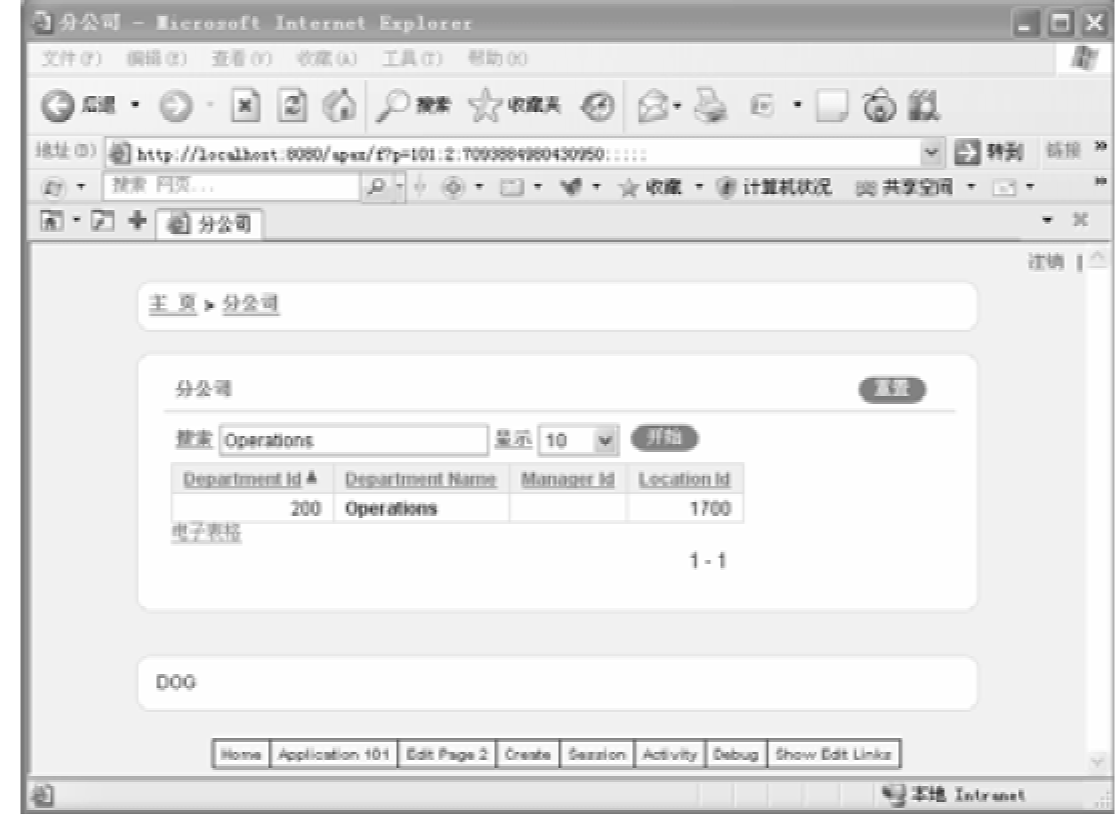


图　20.29

● 重置：当单击“重置”按钮之后，将恢复“显示”的默认设置（15 行），如图 20.31 所示。



图　20.30



图　20.31

● 行列表：用于选择在网页中要看到哪一组记录行，如图 20.32 所示。

● Previous 和 Next：用于快速地显示前一页和后一页的记录集，如图 20.33 所示。

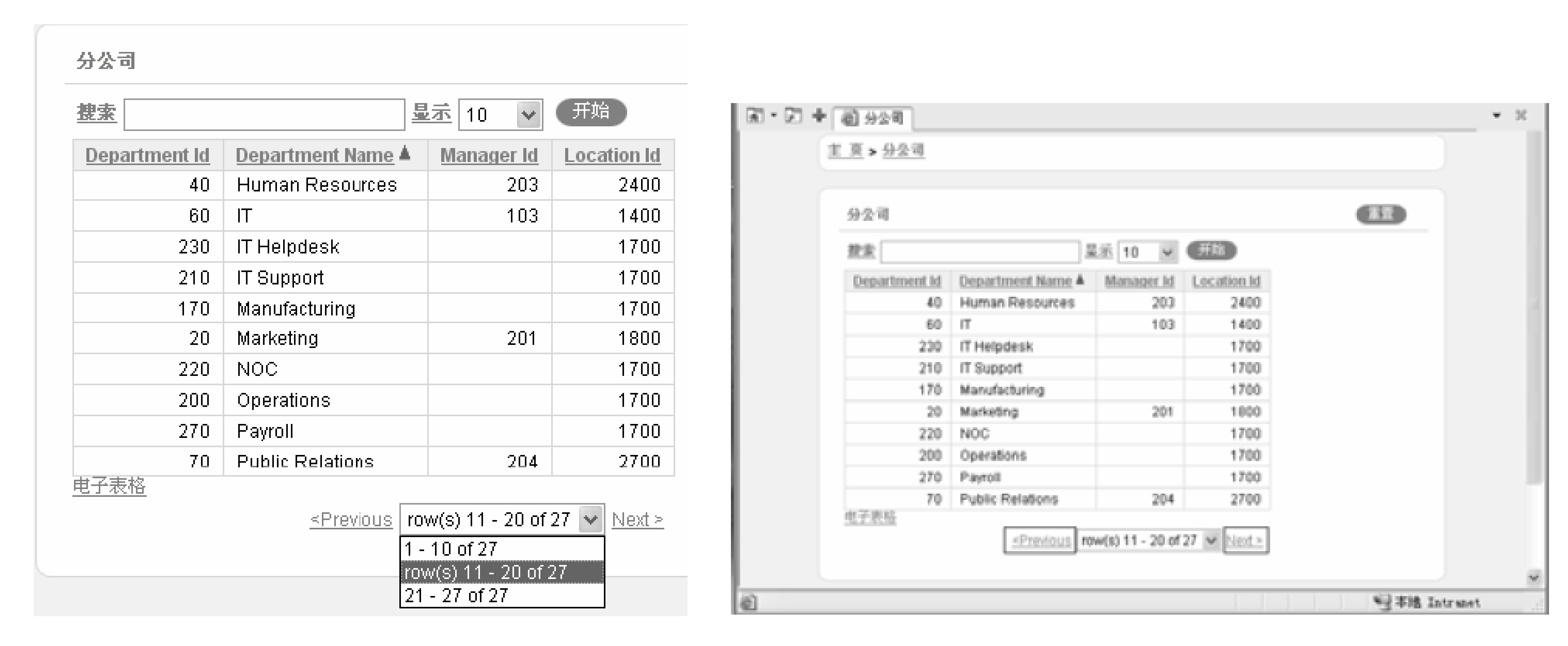


图　20.32　　　　　　　　　图　20.33

● 电子表格：用于将数据存储为电子表格（CSV）文件，如图 20.34 所示。当单击“电子表格”超链接之后，就会出现如图 20.35 所示的页面。



图　20.34



图　20.35

如果您单击“打开”按钮就会将所显示的数据转换成 Microsoft Office Excel 的格式，如图 20.36 所示。您也可以单击“保存”按钮将数据直接存储为 Microsoft Office Excel 文件。原来这么容易就可以将 Oracle 的数据转换成微软系统的格式，并不需要像网上的一些“大虾们”介绍的那样，要进行相当复杂的系统配置。现在您是不是觉得自己瞬间也变成了一个“超级大虾”了？

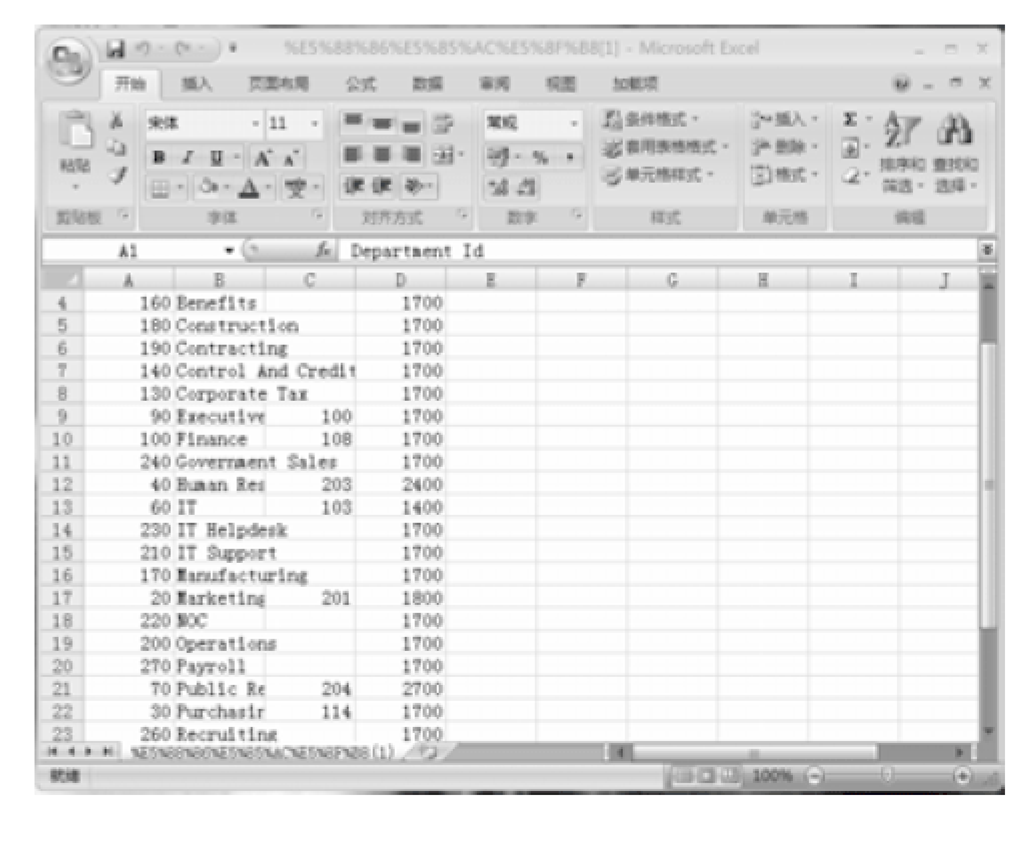


图　20.36

20.3　使用网络浏览器启动应用程序主页

使用网络浏览器启动应用程序主页的具体操作步骤如下：

（1）首先退回到您所创建的应用程序的主页，然后复制地址栏中的地址，如图 20.37 所示。

（2）启动网络浏览器并将刚才复制的地址粘贴到浏览器的地址栏中，然后单击“转到”按钮就进入如图 20.38 所示的登录界面。在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”，然后单击“登录”按钮。



图　20.37

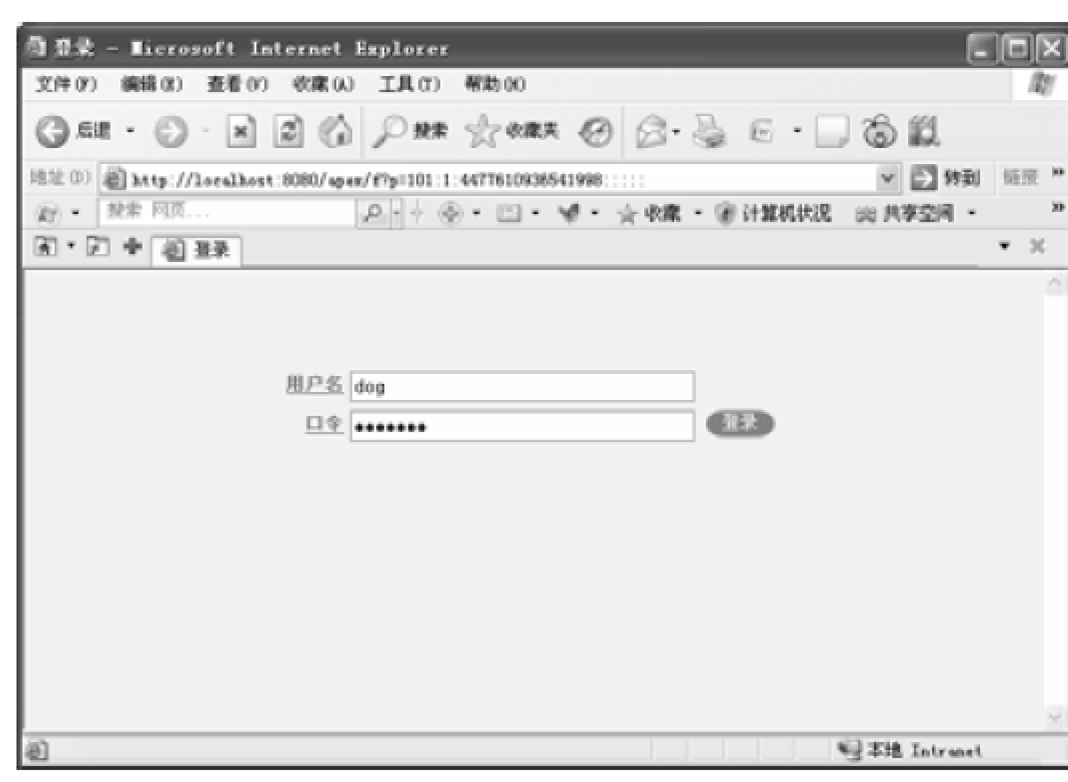


图　20.38

（3）之后将显示您的应用程序主页，如图 20.39 所示。细心的读者可能已经注意到了，就是在图 20.38 和图 20.39 中都没有开发工具栏。当开发的应用系统经过测试后，就会部署到真实的生产环境中，此时普通用户都是以这种方式登录后使用应用程序的，因为他们不需要、也不应该修改系统。

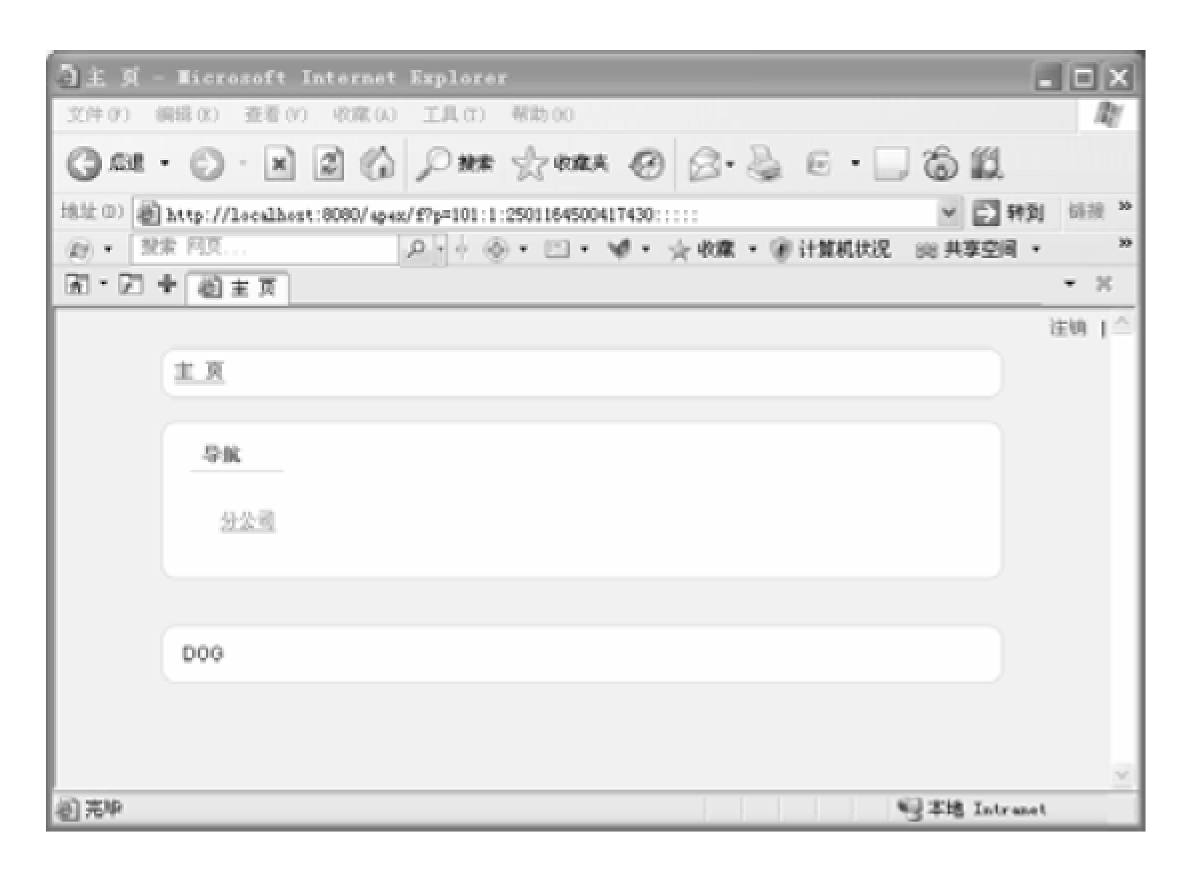


图　20.39

当用户运行创建应用程序向导时，向导就会基于用户所做的选择创建表单和报表，其实是 Application Express 在后台生成了为用户从数据库中抽取所需数据的查询语句。

20.4　修改分公司报表

在本节中，您将修改产生分公司报表的查询语句，其信息同样来自数据库中 HR 用户的 DEPARTMENTS 表。

当老板看过了您刚创建的网页之后，对您的工作非常满意，因为有软件开发商的经理曾告知这么大规模的开发工作至少需要几个程序员开发数周才能完成。现在老板觉得您也是难得的 IT 人才了。不过作为一位著名的民营企业家，她看到网页中那些如 Department id 和 Department Name 之类的就觉得眼晕。因此，她要求您赶快将所有显示的列名都改成比武大郎烧饼的历史还要悠久的中文，同时她还要求您加上每个分公司或部门的人数、总工资等以帮助经理和商业智能人员的分析和决策工作。以下就是具体的操作步骤：

（1）单击页面最底部的开发工具栏中的 Edit Page 2 超链接，如图 20.40 所示。之后就会进入“页 2”的页定义界面，如图 20.41 所示。页是创建应用程序的基本构件，它包含选项卡、列表、按钮、项和区域等用户接口元素。如果要查看属于您的应用程序的页的定义，您就需要使用页定义页面。

（2）在“页呈现”区域的区域栏中单击“分公司”超链接，如图 20.41 所示，将进入编辑区界面，如图 20.42 所示。



Department Id	Department Name	Manager Id	Location Id
40	Human Resources	203	2400
60	IT	103	1400
230	IT Helpdesk		1700
210	IT Support		1700
170	Manufacturing		1700
20	Marketing	201	1800
220	NOC		1700
200	Operations		1700
270	Payroll		1700
70	Public Relations	204	2700

图　20.40

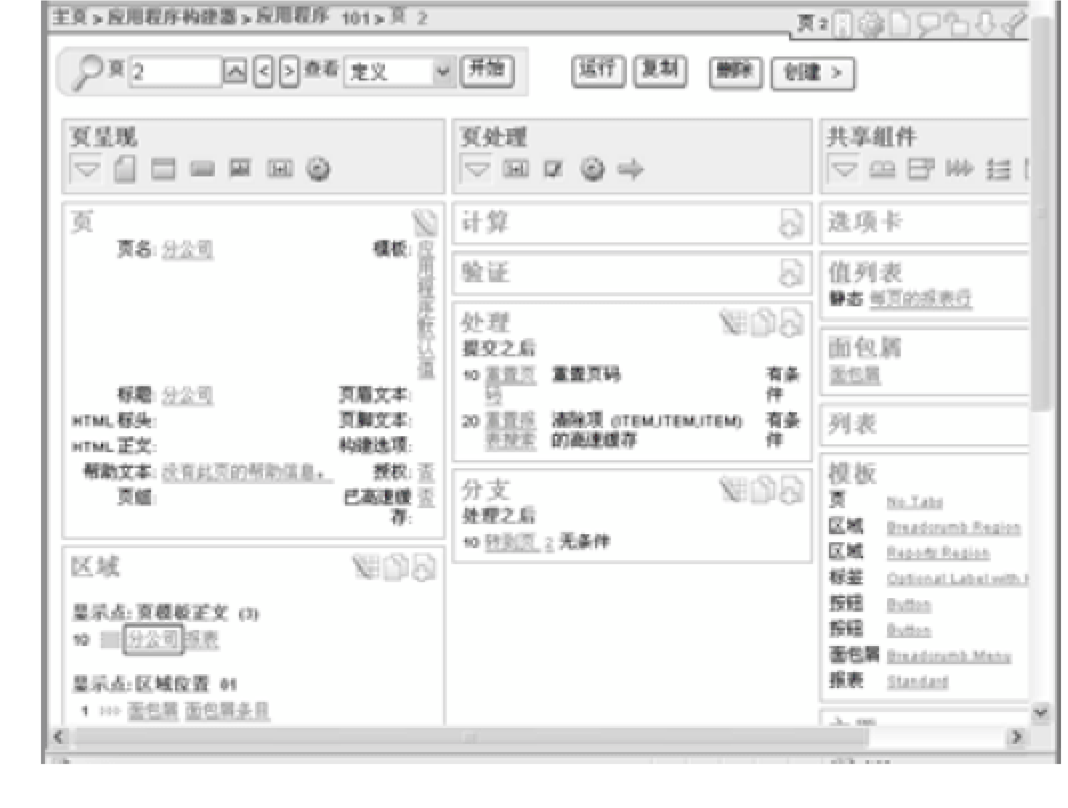


图　20.41

（3）向下滚动到“源”区域，就可以看到“区域源”列表框中默认的 SQL 语句，这是 Oracle Application Express 根据您的选择生成的，如图 20.42 所示。

（4）使用如下的 SQL 语句取代“区域源”列表框中原来的代码（为了方便没有 SQL 知识的读者学习，将这段代码存储在本书的光盘上的 ch20_1.sql 文件中，读者可以直接复制、粘贴到“区域源”列表框中，如图 20.43 所示）。接着单击“应用更改”按钮。

```
SELECT d.department_id "分公司号",
d.department_name "分公司名",
count(e2.employee_id) "员工人数",
min(e2.salary) "最低工资",
```

```
max(e2.salary) "最高工资",
sum(e2.salary) "总工资",
substr(e.first_name,1,1)||'. '|| e.last_name "经理名",
c.country_name "所在地"
FROM departments d,
employees e,
locations l,
countries c,
employees e2
WHERE d.manager_id = e.employee_id
AND d.location_id = l.location_id
AND d.department_id = e2.department_id
AND l.country_id = c.country_id
AND
instr(upper(d.department_name),upper(nvl(:P2_REPORT_SEARCH,d.department
_name))) > 0
GROUP BY d.department_id,
d.department_name,
substr(e.first_name,1,1)||'. '||e.last_name, c.country_name
```



图　20.42

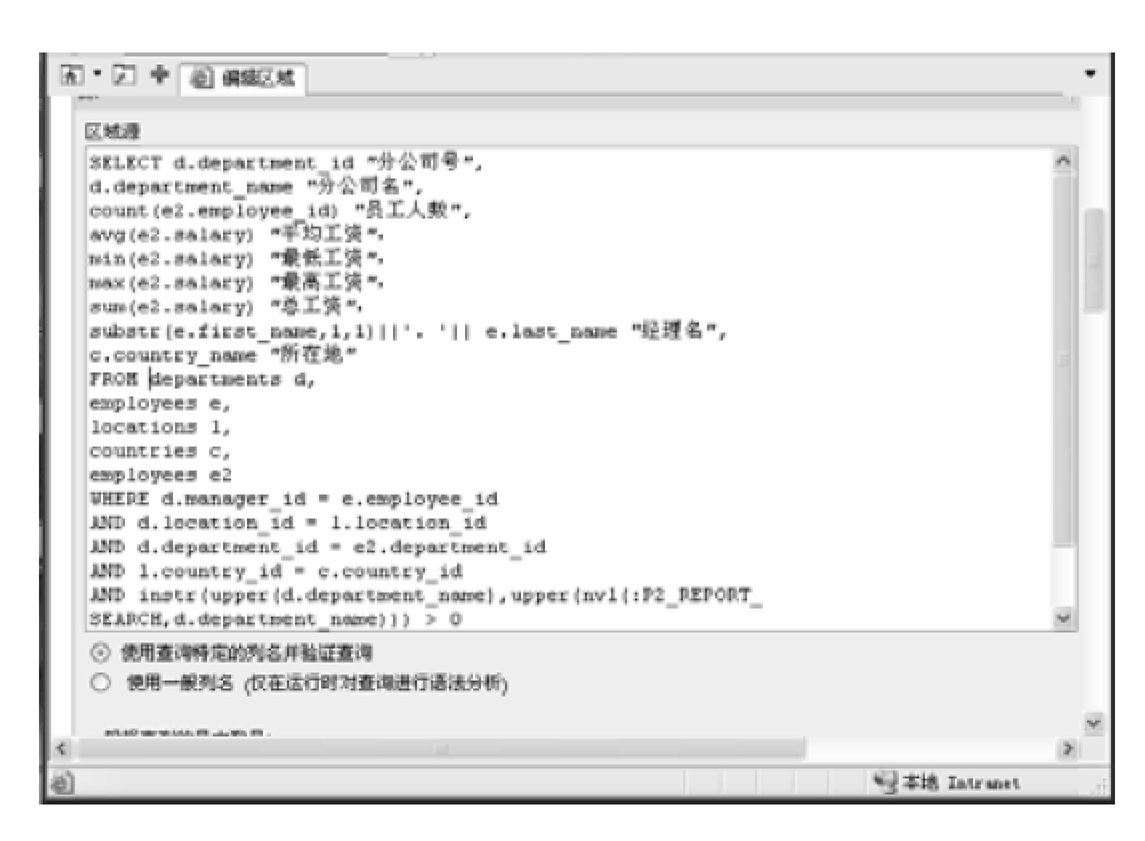


图　20.43

（5）然后在“页定义”页单击“运行页”图标，如图 20.44 所示。

（6）最后将显示您的老板所需的分公司信息，如图 20.45 所示。

☞ 指点迷津：

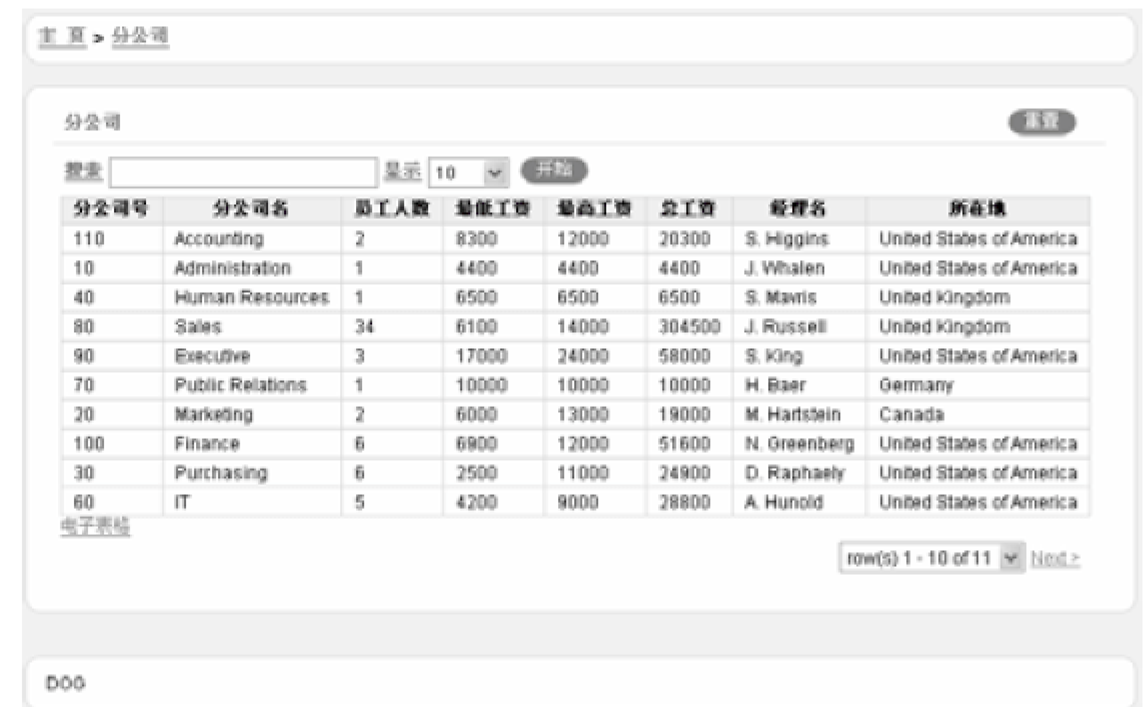
在数据仓库或决策支持系统中，一般情况下，如果用户让您显示平均值（avg）或总和（sum），您最好将数据的行数一起求出。因为只有知道了参与平均或总和的数量，这个平均或总和才有意义。同时，最好将最小值和最大值也一同求出，因为这样管理者或商业智能人员在分析数据时就有了很好的参照点。

另外，在数据量很大时，以上的分组函数操作会严重影响数据库系统的效率。因此，一些有经验的系统开发或设计人员为了提高系统的效率，将这些由分组函数操作所获得的

值存入单独的表中（也叫快照，Oracle 现在的版本也叫物化视图），在接下来的查询中就直接查询这个表而不用进行耗时的分组计算了。



图 20.44



分公司号	分公司名	员工人数	最低工资	最高工资	总工资	经理名	所在地
110	Accounting	2	8300	12000	20300	S. Higgins	United States of America
10	Administration	1	4400	4400	4400	J. Whalen	United States of America
40	Human Resources	1	6500	6500	6500	S. Mavris	United Kingdom
80	Sales	34	6100	14000	304500	J. Russell	United Kingdom
90	Executive	3	17000	24000	58000	S. King	United States of America
70	Public Relations	1	10000	10000	10000	H. Baer	Germany
20	Marketing	2	6000	13000	19000	M. Hartstein	Canada
100	Finance	6	6900	12000	51600	N. Greenberg	United States of America
30	Purchasing	6	2500	11000	24900	D. Raphaely	United States of America
60	IT	5	4200	9000	28800	A. Hunold	United States of America

图 20.45

20.5 添加员工报表和表单

老板看到您刚开发的应用系统已经能以网页的形式实时地展示一些对决策者相当重要的信息之后，她立即决定将这个应用系统放在公司的内网上以提高决策者的决策水平和速度。

现在老板对您也更有信心了，本来最初公司的高级管理层准备将这一应用系统的设计和开发承包给某一大型的软件公司。因为要价太高（相当于数百万个驴肉火烧），老板迟迟下不了决心。现在她真是喜出望外，因为她居然稀里糊涂地招进来一个财神爷，现在您的市价已经至少等于几百万个驴肉火烧了。为了进一步方便决策者和商业智能人员的工作，她让您在网页中加入员工的信息，并且允许经理们在必要时修改这些信息。这实际上是在网页中添加一个员工的报表和表单，以下就是操作的具体步骤：

（1）在主页中单击开发工具栏（在该页的最底部）上的 Application 101 超链接，如图 20.46 所示。

（2）之后将出现应用程序 101 的主页，单击“创建页”按钮，如图 20.47 所示。



图 20.46



图 20.47

（3）对于页，首先选中“表单”单选按钮并单击“下一步”按钮，如图 20.48 所示；然后选中“基于表的表单和报表”单选按钮（这个选项将创建两页，它们是基于同一个表或视图的报表和表单）；最后单击“下一步”按钮，如图 20.49 所示。

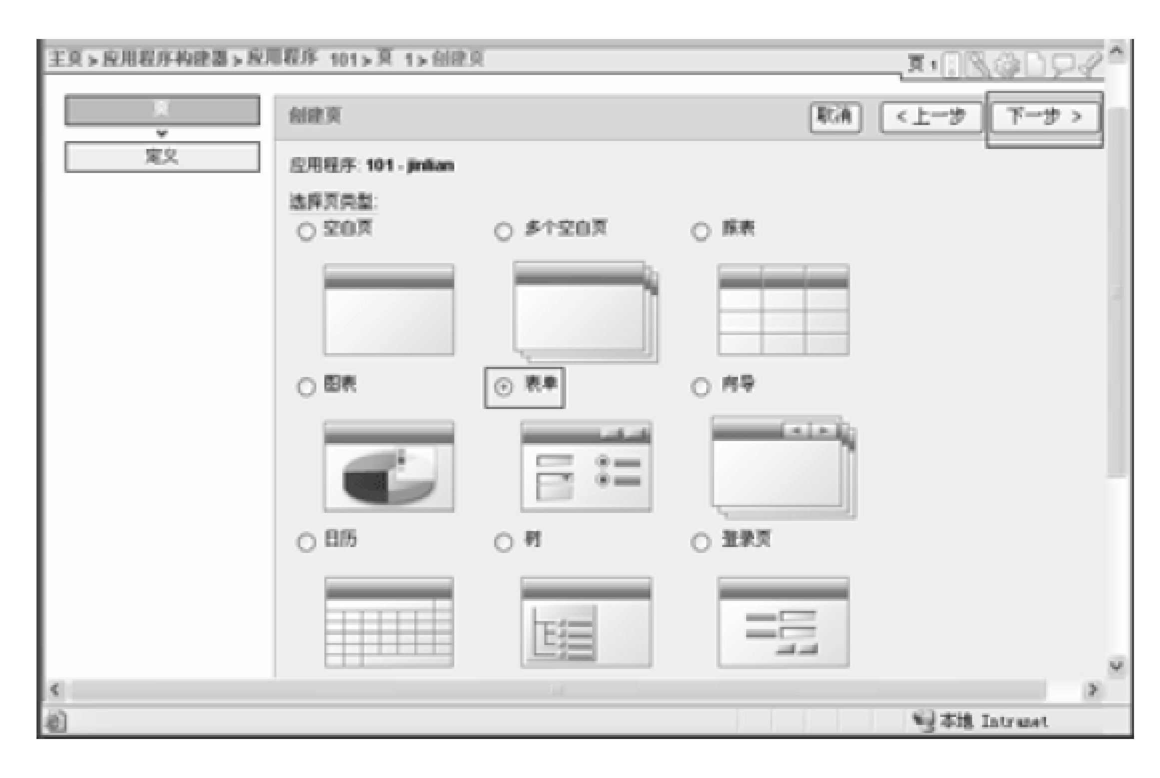


图　20.48



图　20.49

（4）对于标识表或视图，首先在“表/视图所有者”下拉列表框中接受默认用户 HR 并单击“下一步”按钮，如图 20.50 所示；然后在“表/视图名”文本框中选择 EMPLOYEES 并单击“下一步”按钮，如图 20.51 所示。

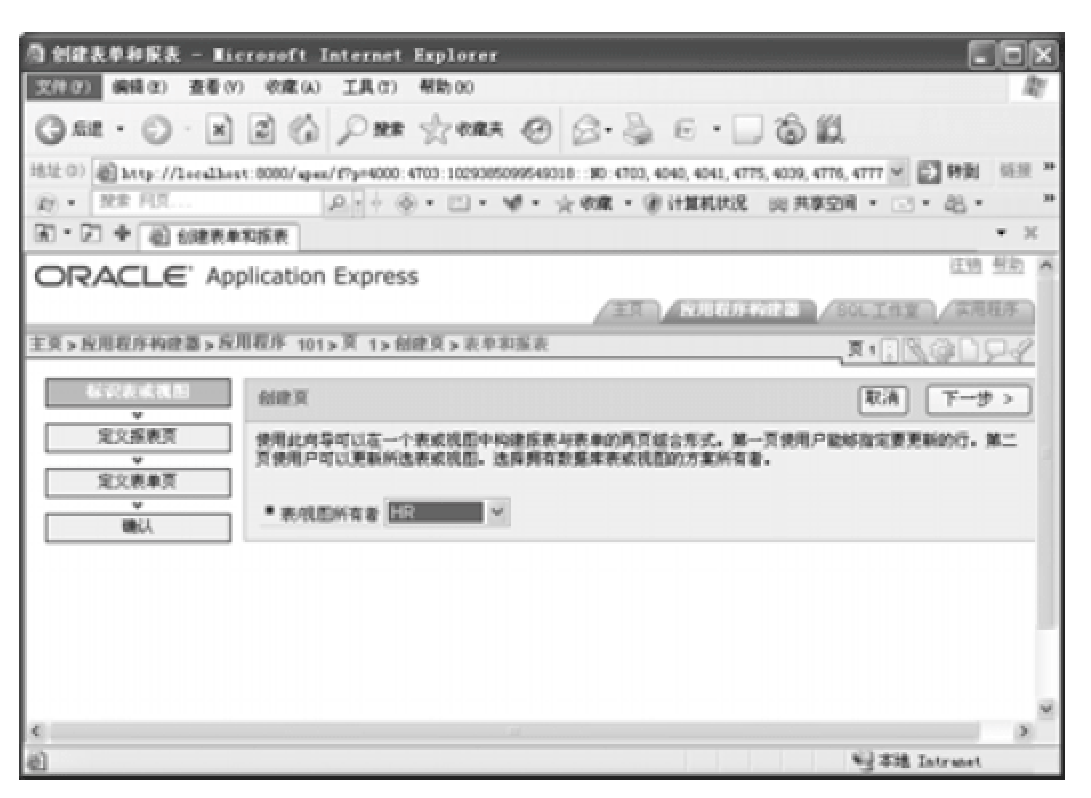


图　20.50

图　20.51

（5）对于定义报表页，在“实施”下拉列表框中选择“经典”选项，将“页名”修改为“人才荟萃”，将“区域标题”也修改成“人才荟萃”；在“面包屑”下拉列表框中选择“面包屑”选项，之后就会出现“创建面包屑条目”区域。单击“主页”超链接就选择了父条目，主页出现在父条目的域中，最后单击“下一步”按钮，如图 20.52 所示。

（6）接受默认选中的“不使用选项卡”单选按钮并单击“下一步”按钮，如图 20.53 所示。

（7）按住 Ctrl 键并选择 EMPLOYEE_ID、FIRST_NAME、LAST_NAME、HIRE_DATE、JOB_ID、SALARY 和 COMMISSION_PCT 列（您所选的列将出现在报表页上），单击“下一步”按钮，如图 20.54 所示。

（8）接受默认设置并单击“下一步”按钮，如图 20.55 所示。到此为止，您已经完成了报表页的定义，接下来就要定义表单页了。



图 20.52

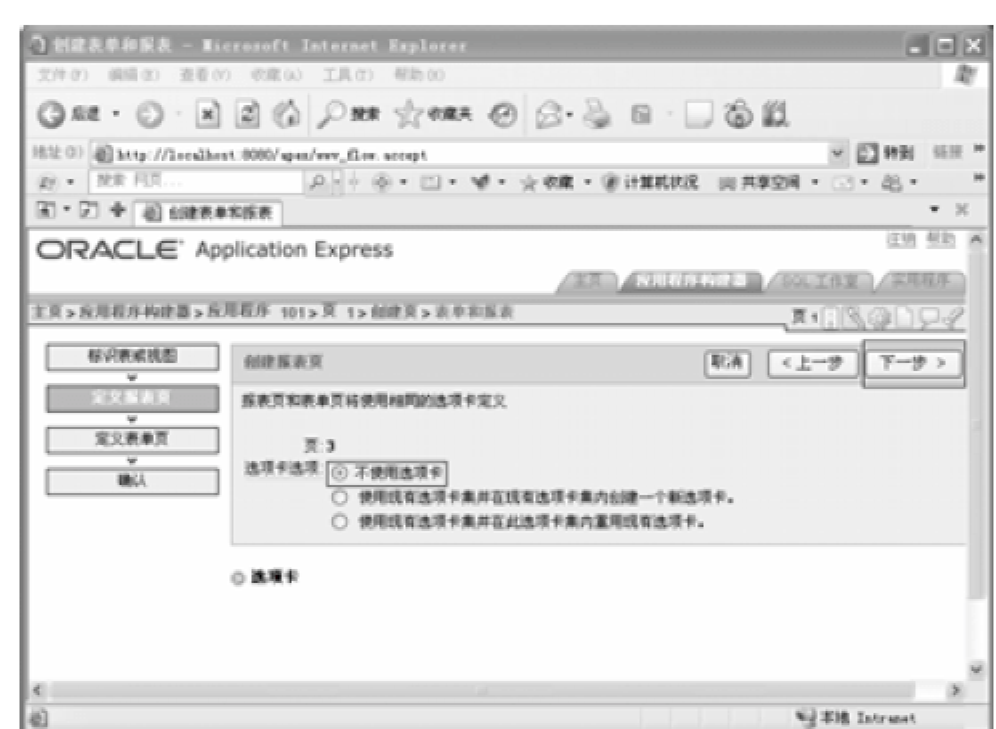


图 20.53



图 20.54

图 20.55

（9）对于定义表单页，在“页名”文本框中输入“创建/编辑 人才信息”，在“区域标题”文本框中也输入“创建/编辑 人才信息”，在“条目名”文本框中再次输入“创建/编辑 人才信息”，然后单击“下一步”按钮，如图 20.56 所示。

（10）主键接受默认设置并单击“下一步”按钮，如图 20.57 所示。



图 20.56



图 20.57

（11）在“定义主键列的源”栏中接受默认选中的“现有触发器”单选按钮并单击“下一步”按钮，如图 20.58 所示。

（12）在“选择列”列表框中选择所有的列（这些列将会出现在“创建/编辑员工”表

单中）并单击“下一步”按钮，如图 20.59 所示。



图　20.58



图　20.59

（13）在“标识处理选项”区域接受默认设置（这些选择使用户能够添加、修改和创建员工的记录）并单击“下一步”按钮，如图 20.60 所示。

（14）检查您所定义的所有信息，如果无误则单击“完成”按钮，如图 20.61 所示。



图　20.60



图　20.61

之后就会出现如图 20.62 所示的页面，这样您就完成了所需网页的全部定义。此时如果您退回到“应用程序 101”所在页面，将会发现多了两个页，它们就是您刚创建的“3-人才荟萃”和“创建/编辑 人才信息”页。

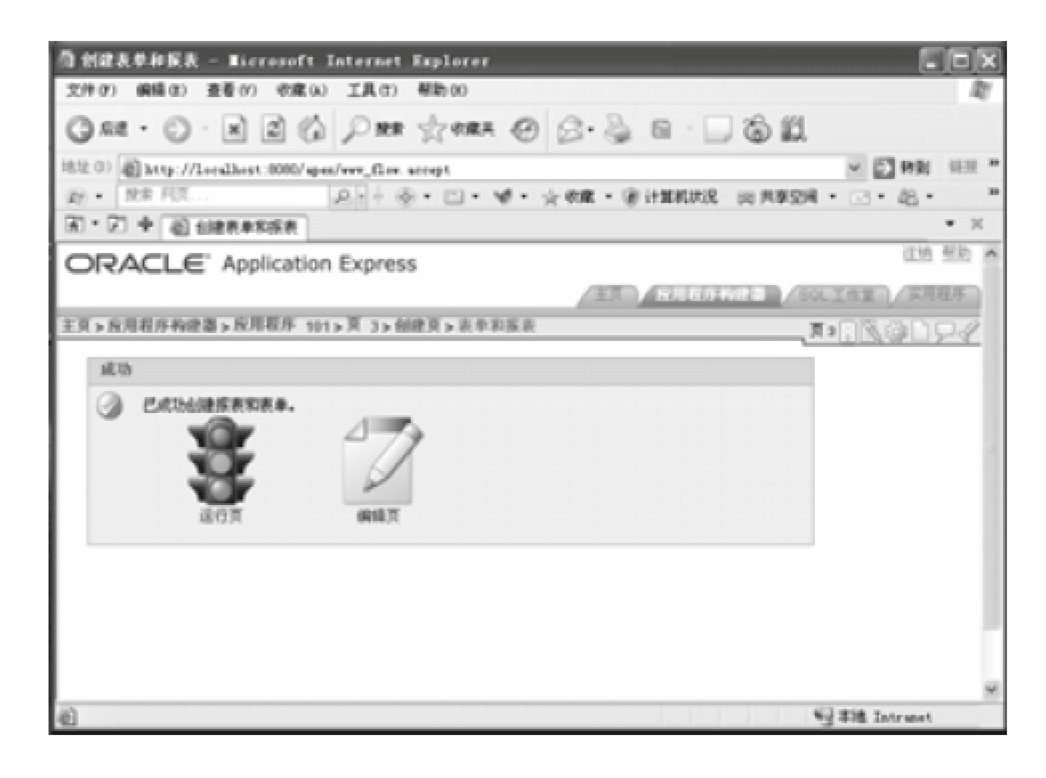


图　20.62

20.6 预览所创建的员工信息网页

要预览刚创建的两个网页，您可以运行当前页（表单和报表），具体操作步骤如下：

（1）在图 20.63 中单击“运行页”图标就会出现如图 20.64 所示的“人才荟萃”页面。

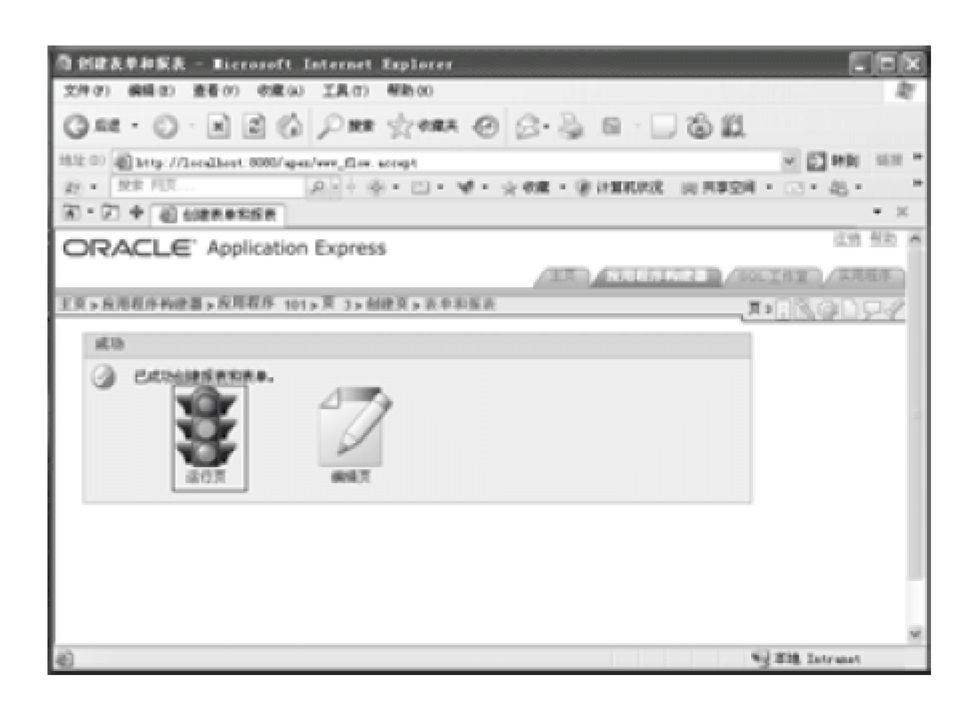


图 20.63

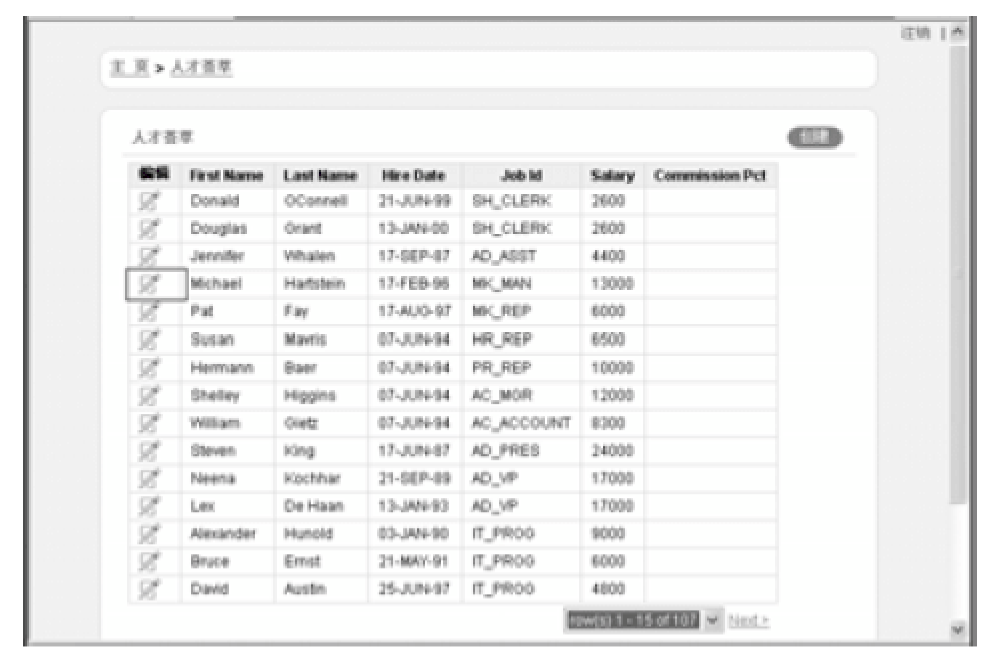


图 20.64

在“人才荟萃”报表中您可以进行如下操作：

- 每一行的最左面都是您在创建时选择的编辑图标，您可以通过单击编辑图标来修改员工（人才荟萃）记录。
- “创建”按钮在该页的右上角，您可以通过单击“创建”按钮将一名员工添加到数据库的表中。
- 员工（人才荟萃）报表包括了您所定义的 6 列。当您选择这些列并提交了更改时，Oracle Application Express 就会在后台创建相应的 SQL 代码并产生所显示的结果。

在您的系统上可能数据的显示顺序有所不同，这无关紧要，请继续后面的操作。

（2）要预览您所创建的表单，单击某一行前面的编辑图标，如 Michael 所在的行，就会出现“创建/编辑 人才信息”表单，如图 20.65 所示。



图 20.65

下面对“创建/编辑 人才信息”表单中的内容进行简单的解释：

- 在该表单中包含了“取消”，“删除”和“应用更改”3 个按钮，用户可以通过它们对数据库表中的数据行进行操作。
- Hire Date 字段处显示了一个日历图标，这是因为该字段所对应的列是日期（DATE）类型。

第21章

编辑 Express 网页

在这一章中，您将创建一个计算员工收入的函数，然后在员工（人才荟萃）报表中加入一列以显示函数的计算值。您还将做一些列的格式化等进一步的编辑工作。

当您向老板演示了你刚创建的“人才荟萃”报表和“创建/编辑 人才信息”表单之后，她对您在这么短的时间内就开发出如此复杂的互联网应用程序感到吃惊，她十分感慨地说：“要是我的员工都以这样的效率做烧饼该有多好啊！”不过那些英文列名仍然令她头疼，她叫您将所有的列名都改成光辉灿烂的古老汉字。

她接着说：“这洋鬼子就是蠢，你看这字，弯弯曲曲的，一点美感也没有，咋卖钱？再看老祖宗给咱们留下的汉字，可以当作画挂在墙上欣赏。前几天我买了一位名人写的几个字就花掉我几千个驴肉火烧。这洋鬼子不但字不行，也不会吃。你看那汉堡包和热狗，简直就是垃圾食品，哪里赶得上咱们的驴肉火烧和狗肉火烧啊！”不管老板怎么说，接下来您还是不得不使用洋人发明的软件开发工具开始完善已经创建的网页。

21.1 创建函数

首先您要创建一个计算员工总收入的函数，具体操作步骤如下：

（1）在应用程序主页中单击开发工具栏（在该页最底部）上的 Application 101 超链接，如图 21.1 所示，之后就会进入 Application 101 所在的页面。

（2）单击标题栏中的“主页”面包屑超链接，如图 21.2 所示。

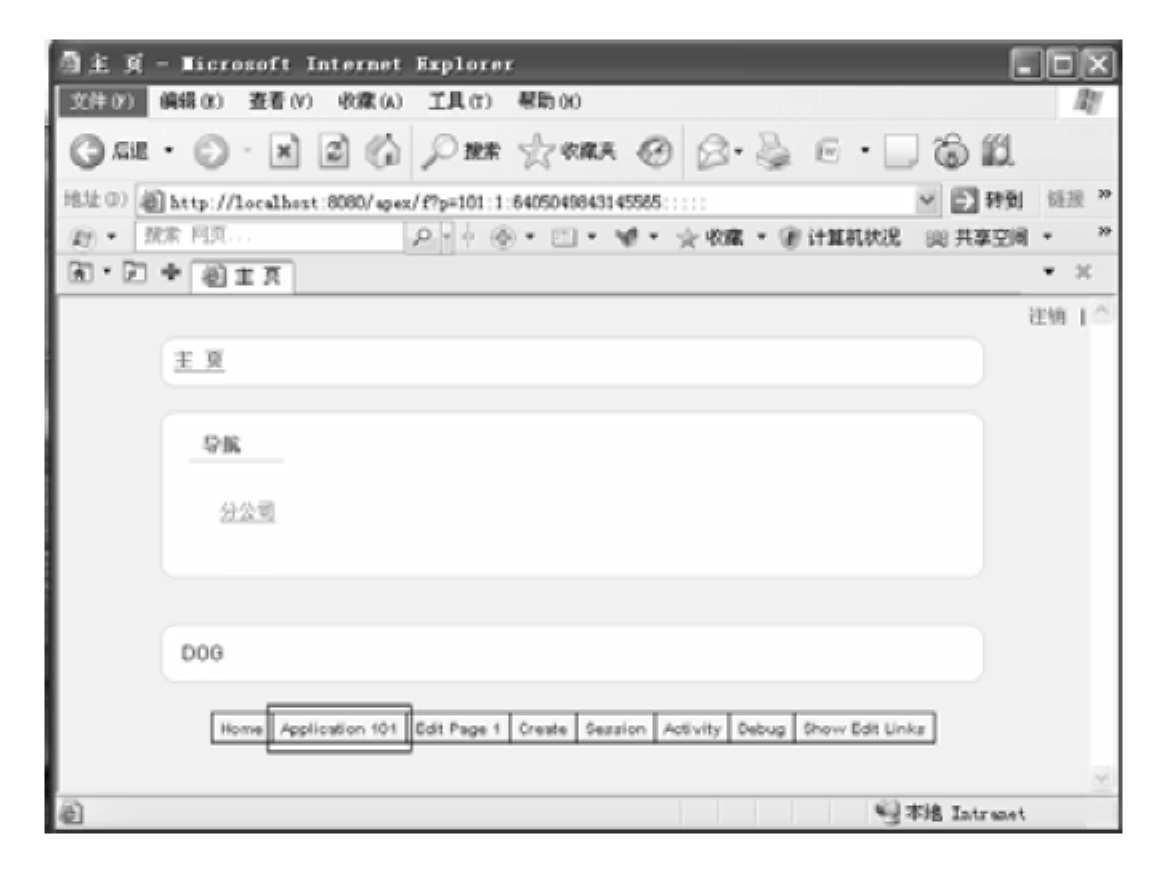


图 21.1



图 21.2

（3）单击“SQL 工作室”图标，如图 21.3 所示。

（4）单击“SQL 命令”图标，如图 21.4 所示。SQL 命令工具提供了一个可以在数据库中运行 SQL 和 PL/SQL 语句的窗口。

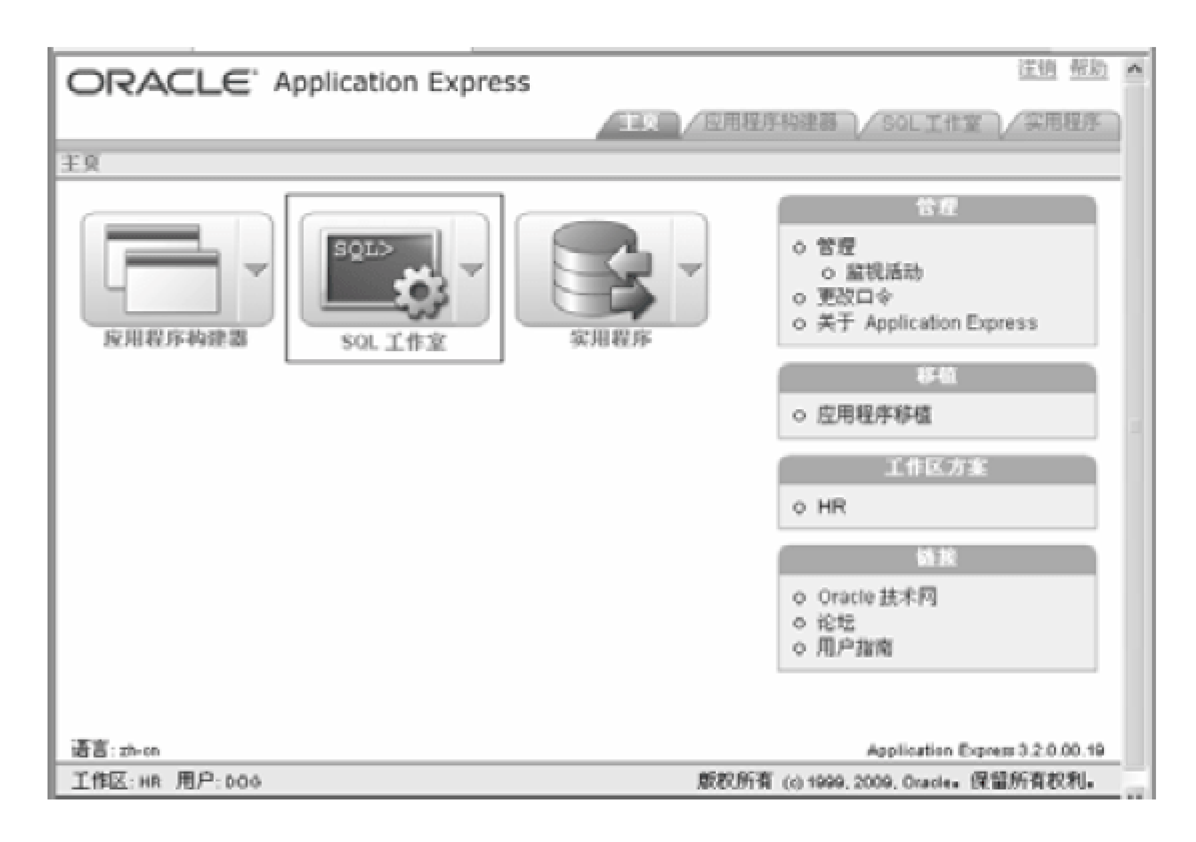


图　21.3

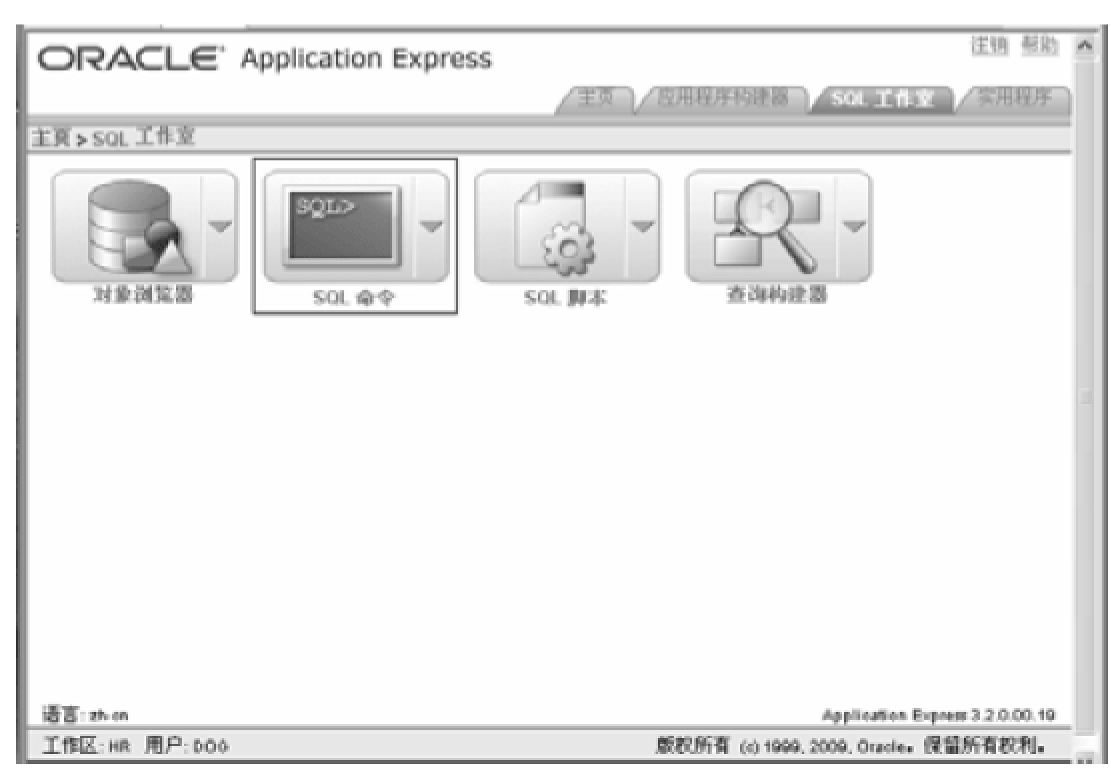


图　21.4

（5）在 SQL 编辑器中输入如图 21.5 所示的创建 PL/SQL 函数的命令（如果读者不熟悉 PL/SQL 程序设计语言，没关系，只要照着输入就行了，因为我们在后面还要给出更简单的方法。另外，在随书的光盘中的 ch21 目录中有一个名为 ch21_1.sql 的文件，其中存有创建这个函数的 PL/SQL 语句，读者可以使用复制或粘贴的方法来完成所需的操作）。

（6）单击“运行”按钮，在 SQL 命令工具的结果部分就将显示“函数已创建。”，如图 21.6 所示。



图　21.5



图　21.6

到此为止，您已经成功地在数据库中创建了一个叫 calc_remuneration 的存储函数，它的功能就是返回一个员工每年的总收入（年薪+佣金）。以后您就可以反复使用这一函数了，其他用户也可以使用它，而且可以同时使用。函数可以非常复杂，这样就可以让有经验的高级开发人员开发一些常用的存储函数，普通用户只要在 SQL 语句中使用它们就行了。用这样的方法可以极大地简化应用系统的开发。

21.2　创建列和修改列名

下面在“人才荟萃”报表中添加一列以显示员工的年收入（就是您刚创建的函数的返回值），同时再将所有的列名改为中文，具体操作步骤如下：

（1）在“SQL 命令”页中单击“主页”面包屑超链接，如图 21.7 所示。

（2）在工作区主页单击“应用程序构建器”图标，如图 21.8 所示。

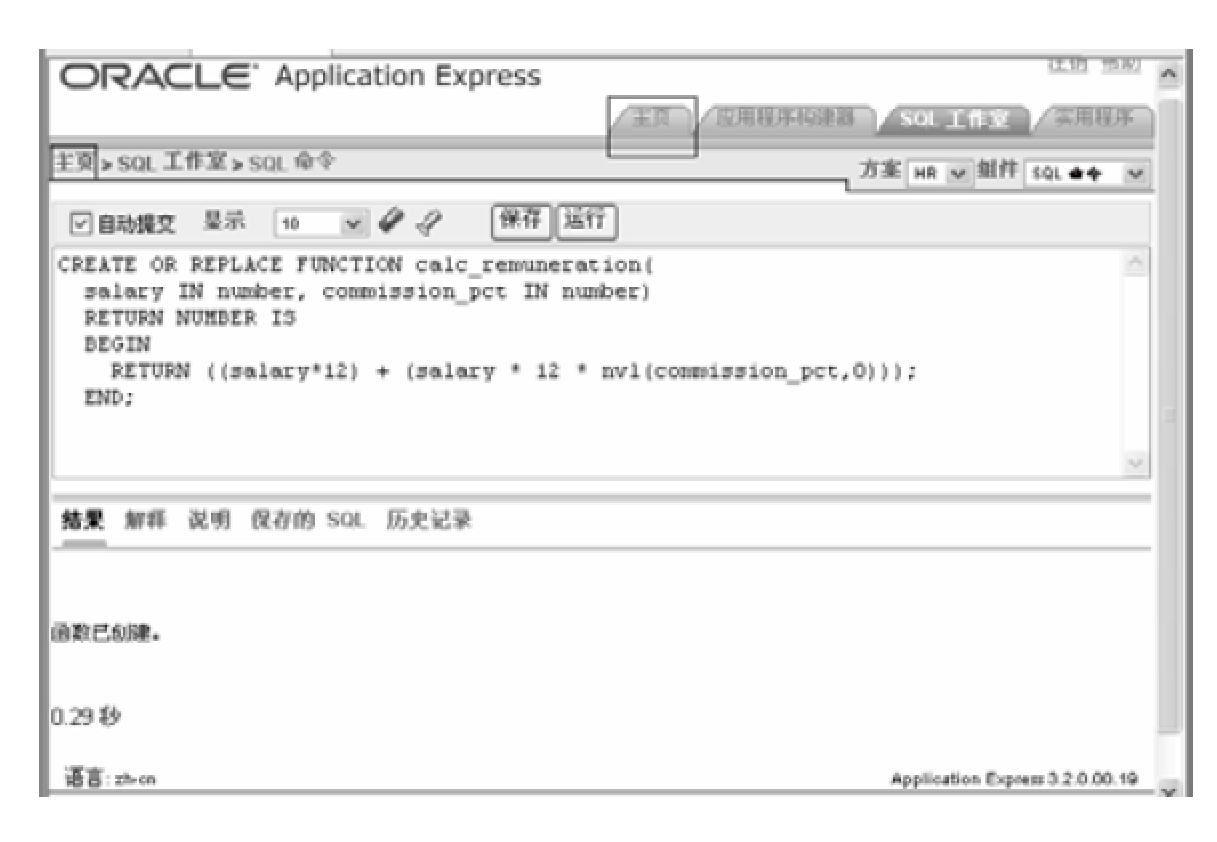


图　21.7



图　21.8

（3）选择应用程序 jinlian-101，如图 21.9 所示。

（4）选择“3-人才荟萃”选项，如图 21.10 所示。



图　21.9



图　21.10

（5）在“区域”栏中单击“人才荟萃”超链接，如图 21.11 所示。

（6）之后将出现区域的定义，向下滚动到“区域源”列表框，就会出现现有的 SQL 语句，如图 21.12 所示。

（7）利用图 21.13 中的代码取代现有的 SQL 代码（在随书的光盘中的 ch21 目录中有一个名为 ch21_2.sql 的文件，其中存有所需的 SQL 语句，读者可以使用复制或粘贴的方法来完成所需的操作）。


图 21.11


图 21.12


图 21.13

☞ **指点迷津：**

在区域源中使用表达式 salary * 12 * (1 + nvl(commission_pct,0))取代函数 calc_remuneration(salary, commission_pct)调用可以得到完全相同的结果。也就是说，读者完全可以不定义 calc_remuneration 函数就可以完成以上的操作。本书使用 PL/SQL 函数的目的是为了说明在 Oracle Application Express 中可以加入复杂的 PL/SQL 函数，以方便开发大型复杂的商业应用系统。

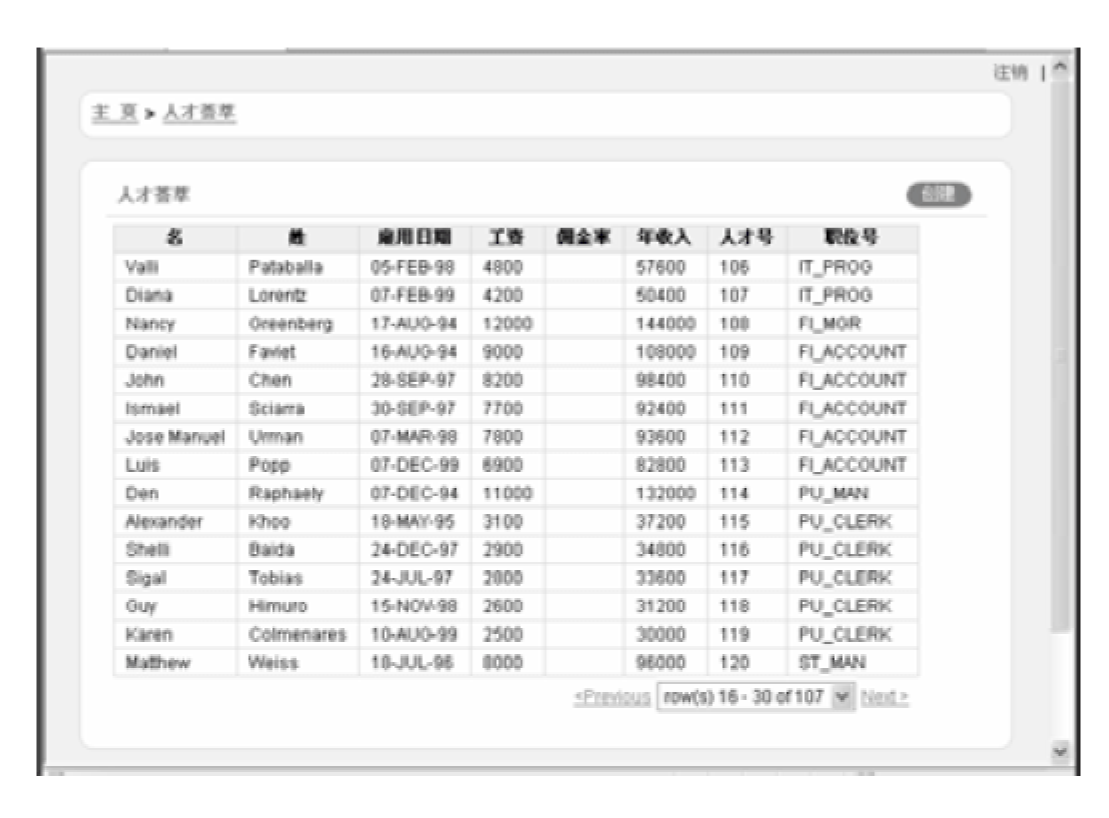
（8）单击“应用更改”按钮，如图 21.14 所示。

（9）单击右上角的“运行页”图标即可预览网页，如图 21.15 所示。


图 21.14


图 21.15

这样就会得到如图 21.16 所示的员工（人才荟萃）报表，而且每一列的名都已经改为了中文显示。



名	姓	雇用日期	工资	佣金率	年收入	人才号	职位号
Valli	Pataballa	05-FEB-98	4800		57600	106	IT_PROG
Diana	Lorentz	07-FEB-99	4200		50400	107	IT_PROG
Nancy	Greenberg	17-AUG-94	12000		144000	108	FI_MGR
Daniel	Faviet	16-AUG-94	9000		108000	109	FI_ACCOUNT
John	Chen	28-SEP-97	8200		98400	110	FI_ACCOUNT
Ismael	Sciarra	30-SEP-97	7700		92400	111	FI_ACCOUNT
Jose Manuel	Urman	07-MAR-98	7800		93600	112	FI_ACCOUNT
Luis	Popp	07-DEC-99	6900		82800	113	FI_ACCOUNT
Den	Raphaely	07-DEC-94	11000		132000	114	PU_MAN
Alexander	Khoo	18-MAY-95	3100		37200	115	PU_CLERK
Shelli	Baida	24-DEC-97	2900		34800	116	PU_CLERK
Sigal	Tobias	24-JUL-97	2800		33600	117	PU_CLERK
Guy	Himuro	15-NOV-98	2600		31200	118	PU_CLERK
Karen	Colmenares	10-AUG-99	2500		30000	119	PU_CLERK
Matthew	Weiss	18-JUL-96	8000		96000	120	ST_MAN

图　21.16

☞ **指点迷津：**

当将列名改成了中文之后，图 21.16 的员工（人才荟萃）报表中最左面的编辑图标消失了。如果您想利用这一界面进行数据库的 DML 操作，可就遇到了麻烦。

21.3　修改列显示格式

下面介绍如何修改数字类型列的显示格式。当老板看到您编辑之后的网页显示时，脸上再一次露出灿烂的笑容。同时，她也问您能不能将“工资”和“年收入”之类的数据加上千位符和小数点，这样更容易浏览。以下就是给数据加上千位符和小数点的具体操作步骤：

（1）在“应用程序 101”页中单击“3-人才荟萃”图标，如图 21.17 所示，将进入“页 3”的定义界面。

（2）在“区域”栏中单击“报表”超链接，如图 21.18 所示，将进入“报表属性”界面。



图　21.17

图　21.18

（3）定位在列属性部分，修改某些列的值和列标题的对齐方式。

（4）在“列对齐”列，在“工资”、“佣金率”和“年收入”下拉列表框中选择“右”选项。

（5）在“标题对齐”列，在“工资”、“佣金率”和“年收入”下拉列表框中选择“居中”选项，如图 21.19 所示。接下来就可以编辑列中值的格式了。

（6）在“列属性”选项卡中单击“工资”左面的编辑图标，如图 21.20 所示，就会出现“列属性:工资”界面。



图 21.19



图 21.20

（7）在“列格式”区域单击“数字/日期格式”文本框后的选择按钮，如图 21.21 所示，将出现选择列表。

（8）在选择列表中选择“￥5,234.10”选项，如图 21.22 所示。



图 21.21

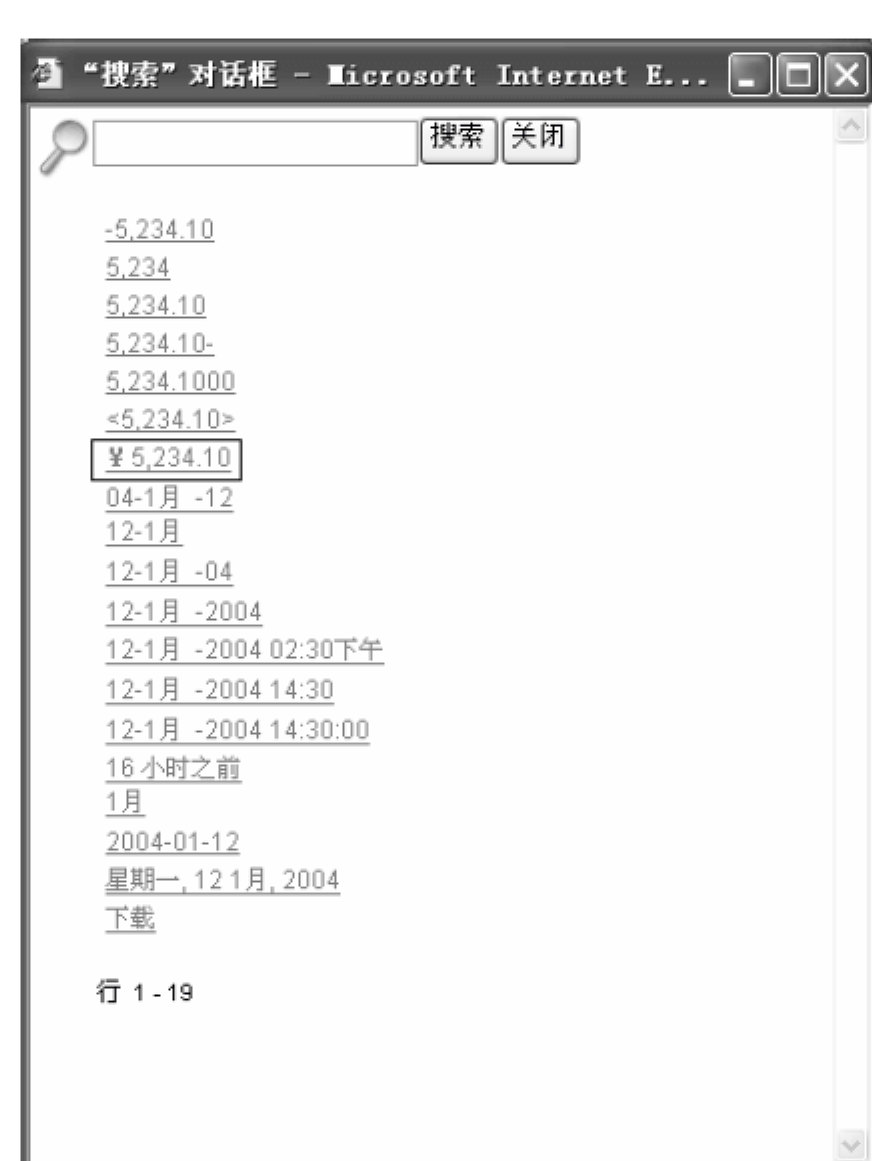


图 21.22

（9）单击“应用更改”按钮，如图 21.23 所示。

（10）对“年收入”列重复以上操作，如图 21.24 所示。



图　21.23



图　21.24

（11）再次单击“应用更改”按钮，如图 21.25 所示。

（12）单击“页 3”右上角的“运行页”图标预览编辑后的网页，如图 21.26 所示。



图　21.25



图　21.26

在图 21.27 中，“工资”和“年收入”的数值之前出现了美元符号，并且数值中还包含了千位符和小数点，同时，也是按您刚定义的对齐方式对齐的。

下面您将使用与上面类似的步骤修改“分公司”网页中数字列的显示格式，具体操作步骤如下：

（1）在“应用程序 101”页面中，单击“2-分公司”图标，如图 21.28 所示。之后将出现页 2 的定义页面。



图　21.27



图　21.28

（2）在“区域”栏中单击“报表”超链接，如图 21.29 所示，将出现“报表属性”页面。

（3）找到列属性部分，调整相应数字列值和列标题的对齐方式，如图 21.30 所示，然后，单击某个数值列的编辑图标就将出现“列属性”页。



图　21.29



图　21.30

（4）在选择列表中选择“<u>￥5,234.10</u>”选项，然后单击“应用更改”按钮，如图 21.31 所示。

（5）重复以上步骤，修改其他数据列的显示格式。

（6）单击“页 2”页右上角的“运行页”图标预览编辑后的网页，如图 21.32 所示。



图　21.31



图　21.32

在图 21.33 中，“最低工资”、“最高工资”和“总工资”的数值之前出现了美元符号，并且数值中还包含了千位符和小数点，同时，也是按您刚定义的对齐方式对齐的。



图　21.33

21.4　以选择列表来显示项的准备工作

由于您刚创建的网页的用户主要是管理人员，他们中的多数人打字速度很慢而且常常出错，因此，老板要求您将经理们经常要操作的数据改成用下拉式菜单显示，这样他们每次操作时只用从菜单中选择已有的数据，可以极大地减少出错的可能性，同时，也可以提高系统的效率。

为了完成老板交给您的重任，可以利用选择列表来编辑在“创建/编辑　人才信息”页上的项（字段）。选择列表是一个下拉式列表，它包括了用户可以在应用程序中使用的所有值。为了以选择列表来显示项，通常要做如下两步的操作：

（1）为所操作的每个项创建一个值列表（LOV）。

（2）引用所创建的对应值列表并以这个选择列表的显示来编辑项。

由于我们将“人才荟萃”报表中的列名都改成了中文，该报表中所有行的编辑图标都已经不见了。为了能继续后面的操作，我们要想办法使这些编辑图标重新出现在这个报表的每一行上。可能有读者想到了，最简单的方法就是将对应 SQL 语句中的中文别名去掉，您不妨可以试一下。以下是主要的操作步骤：

（1）在主页中单击“应用程序构建器”图标，如图 21.34 所示，进入“应用程序构建器”页面。

（2）在“应用程序构建器”页面中单击“jinlian-101”图标，如图 21.35 所示，进入“应用程序 101”页。

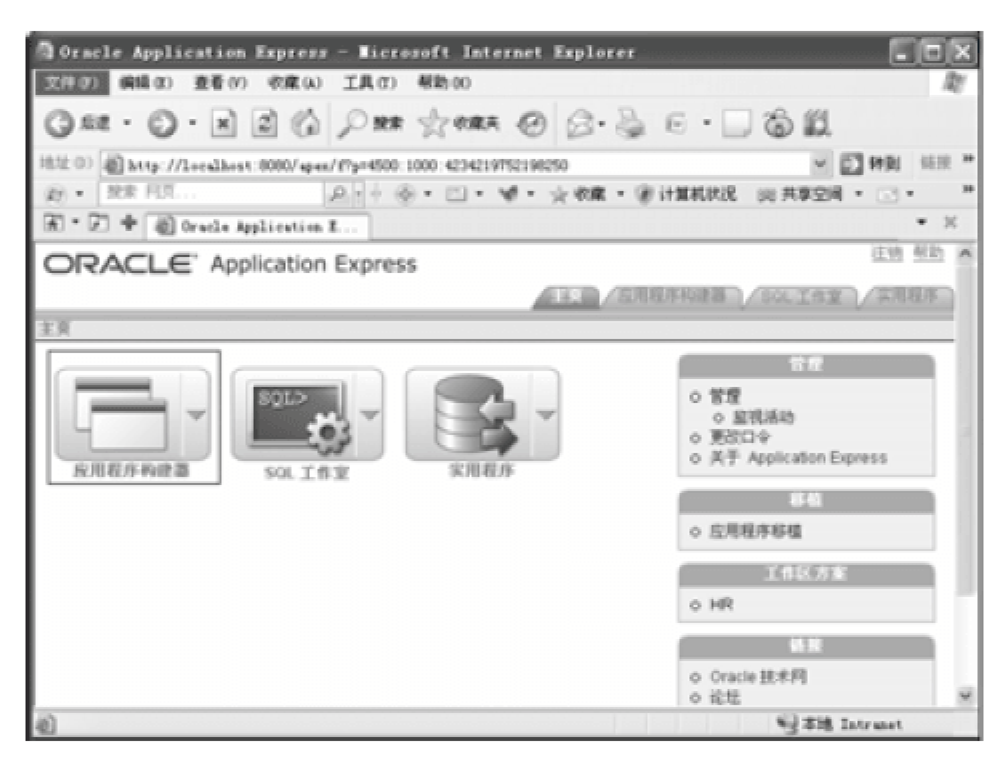


图　21.34

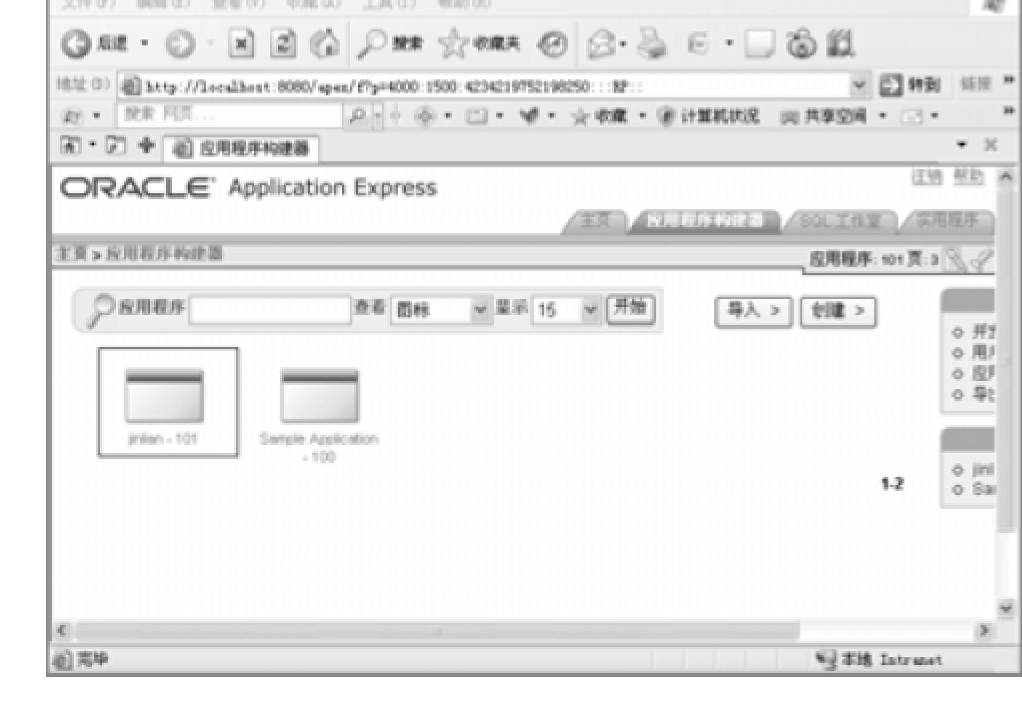


图　21.35

（3）在“应用程序 101”页面中，单击“3-人才荟萃”图标，如图 21.36 所示，进入“3-人才荟萃”页面。

（4）在页 3 的定义页面单击“人才荟萃”超链接，如图 21.37 所示，进入“人才荟萃”页面。

（5）在“人才荟萃”页面找到“区域源”列表框中的 SQL 语句，如图 21.38 所示。

（6）在“人才荟萃”页面将在“区域源”列表框中的 SQL 语句修改成如图 21.39 所示（即去掉中文的别名）。



图 21.36



图 21.37



图 21.38



图 21.39

（7）单击“应用更改”按钮，如图 21.40 所示，将回到“页 3”页面。

（8）单击“运行”图标，如图 21.41 所示，将出现登录页面。



图 21.40



图 21.41

（9）在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”，然后单击“登录”按钮，如图 21.42 所示。

但是，最终您会发现编辑图标并未重新出现，是不是感到有点失望？许多软件在处理中文时经常会出现一些令人感到意外的结果。这不是您的问题，因为这些软件都是西方人开发的，他们不会考虑（也可以说根本就不知道）中文的特殊性。读者在遇到类似的问题

时用不着紧张，可以尝试解决或绕过这些问题。

下面为了简单起见，我们使用最直接的方法，就是先删除页 3 和页 4，之后再重建。为了减少篇幅，这里只给出了页 4 的删除步骤。读者要使用类似步骤自己删除页 3。

（1）在页 4 的“页定义”页面中单击“删除”按钮，如图 21.43 所示。



图　21.42

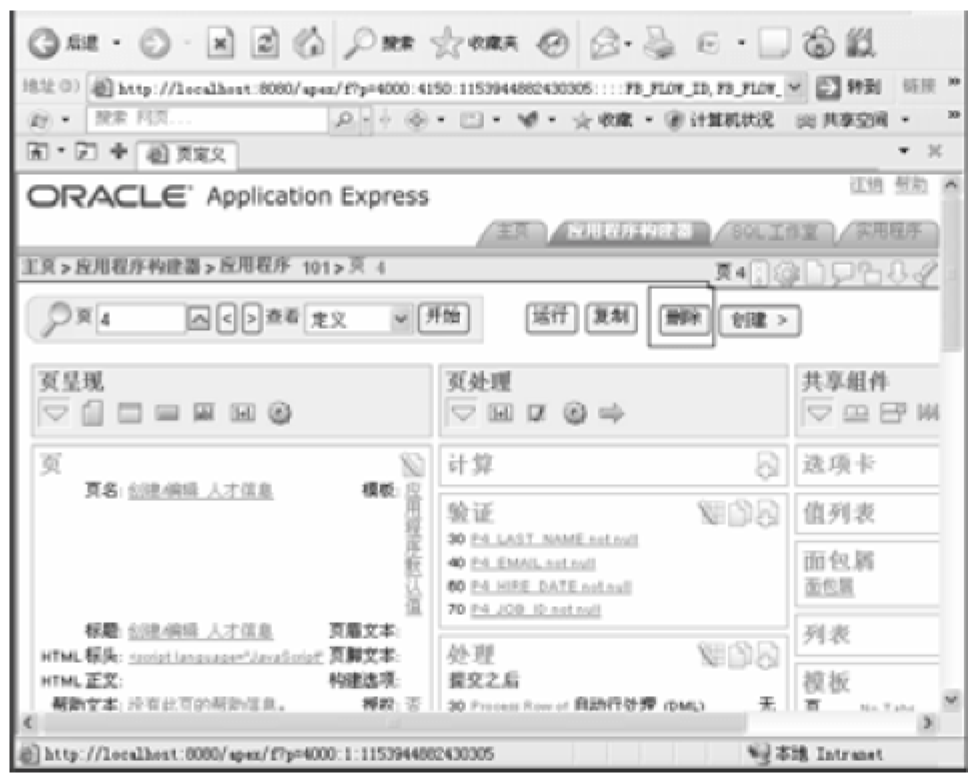


图　21.43

（2）在“确认删除”页中单击“永久删除页”按钮，如图 21.44 所示。

（3）当删除了页 3 和页 4 之后，单击面包屑的“应用程序”超链接，如图 21.45 所示。

（4）当出现如图 21.46 的页面时，就表示您已经成功地删除了页 3 和页 4。

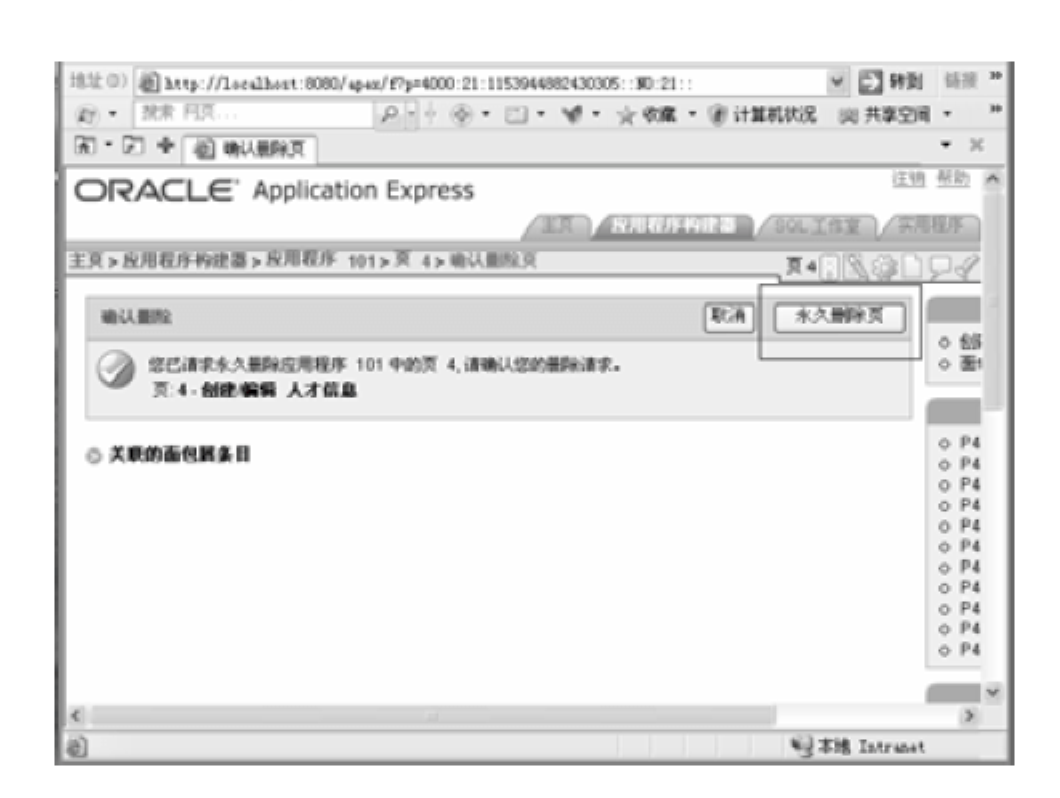


图　21.44

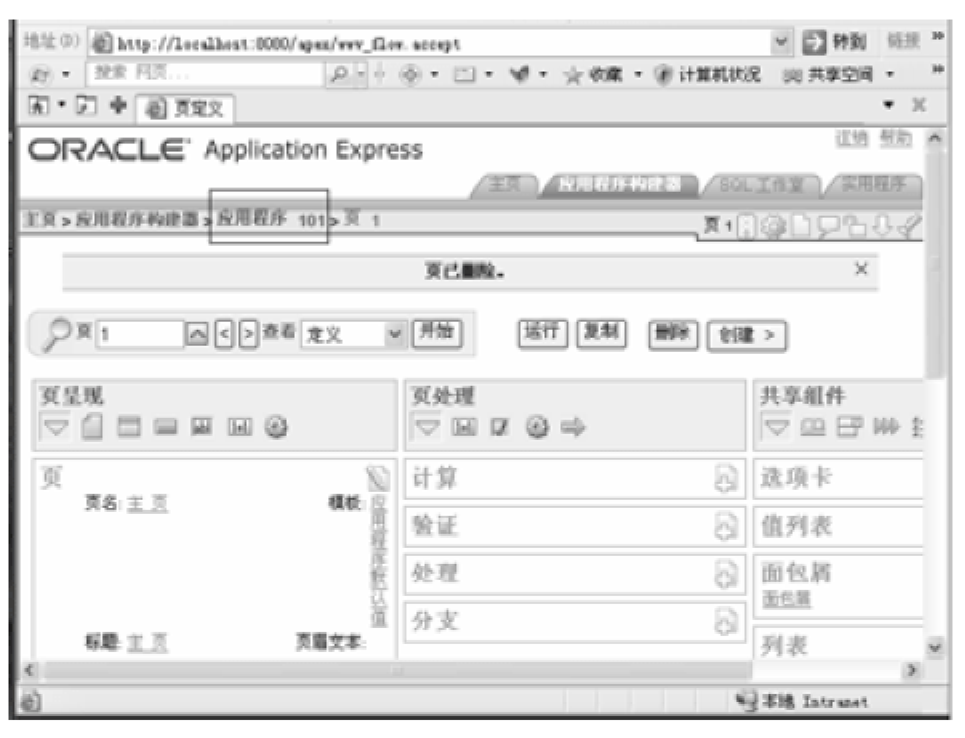


图　21.45



图　21.46

现在读者就可以使用第 20.5 节的方法重新加入这两页，并使用第 21.2 节的方法加入名为 INCOME（收入）的一列。也可以使用第 21.3 节的方法修改数字类型列的显示格式，主要操作步骤如下：

☞ 指点迷津：

在重新加入这两页时最好将页号改为原来的 3 和 4，而不要使用 Express 自动产生的页号，以方便后面的操作。

（1）在区域源的 SQL 查询语句中加入“calc_remuneration(salary, commission_pct) INCOME”，如图 21.47 所示。

（2）单击“应用更改”按钮，如图 21.48 所示。



图 21.47

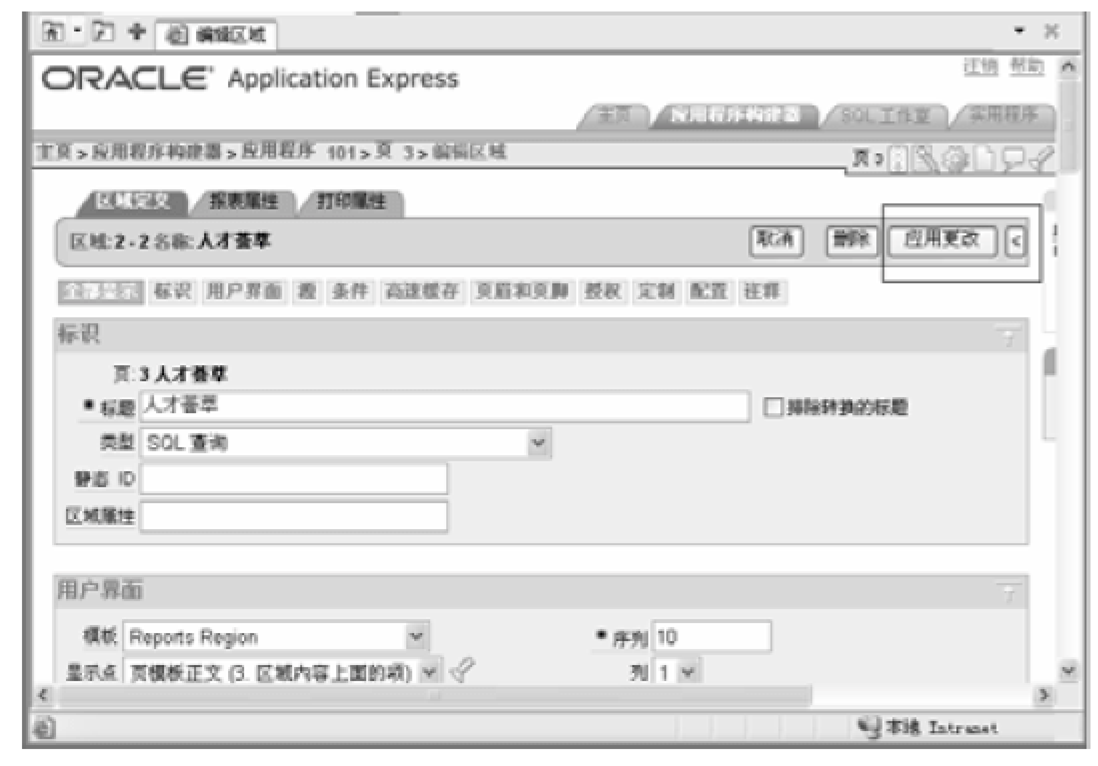


图 21.48

（3）单击“运行页”图标，如图 21.49 所示。

（4）打开“登录”页面，在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”，然后单击“登录”按钮，如图 21.50 所示。



图 21.49

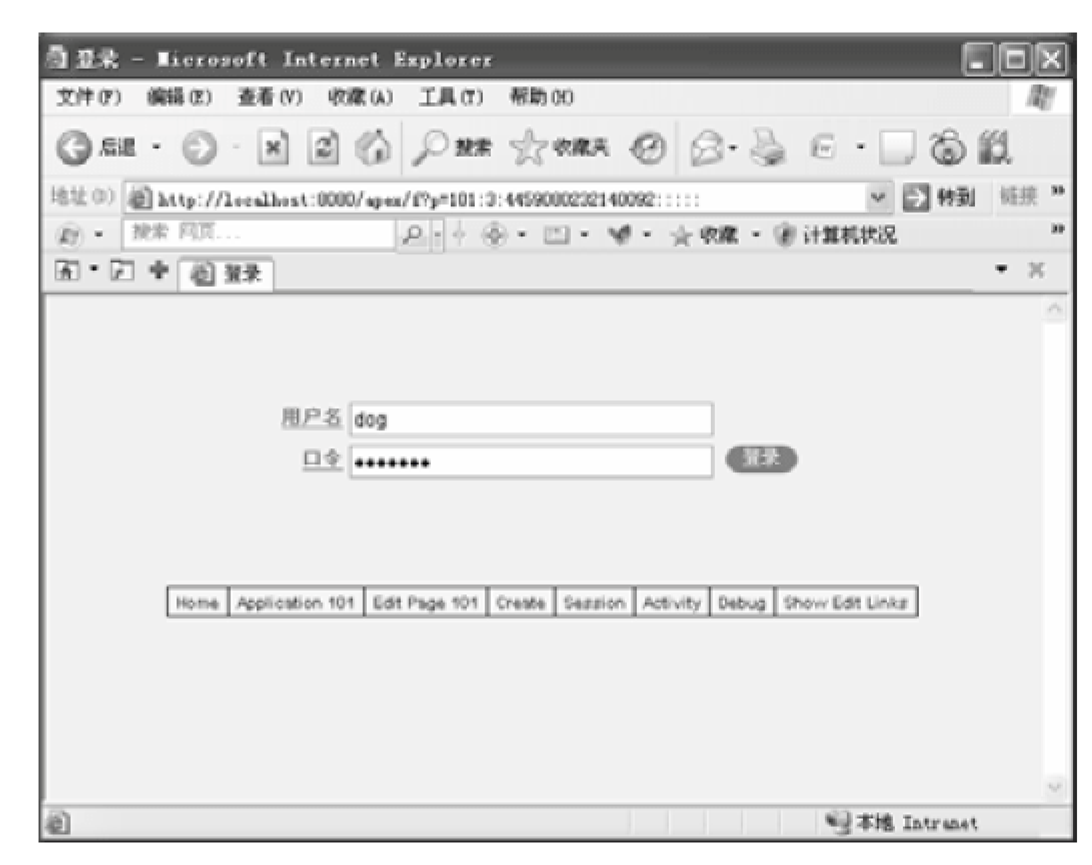
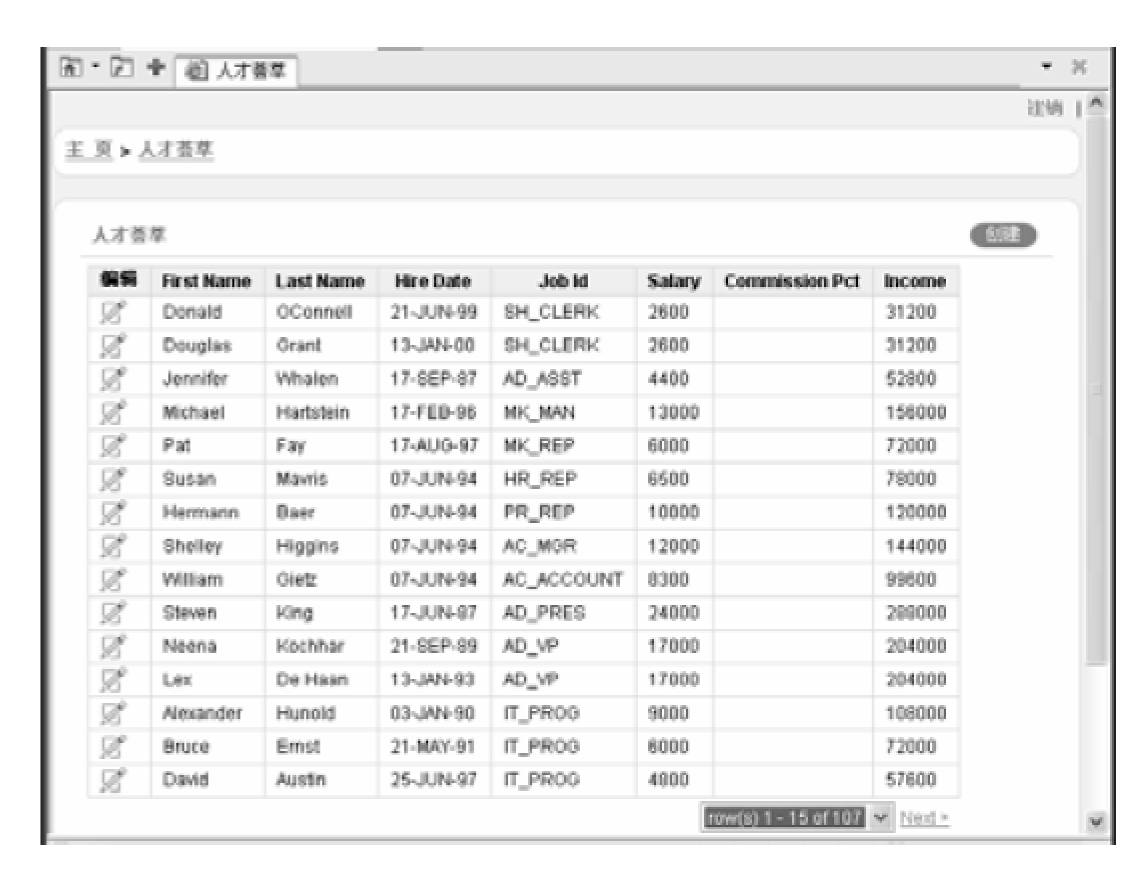


图 21.50

（5）之后就会出现如图 21.51 的页面。终于在每一数据行的最左面都出现了编辑图标，真不容易。中文这一为我们留下了数不胜数的千古绝句的优美语言在现代的高科技应用中

却遇到了麻烦。此时，可以使用第 21.3 节的方法修改数字类型列的显示格式（读者也可以不修改格式）。

（6）最后将出现如图 21.52 所示的页面，接下来就可以创建值列表（LOV）了。



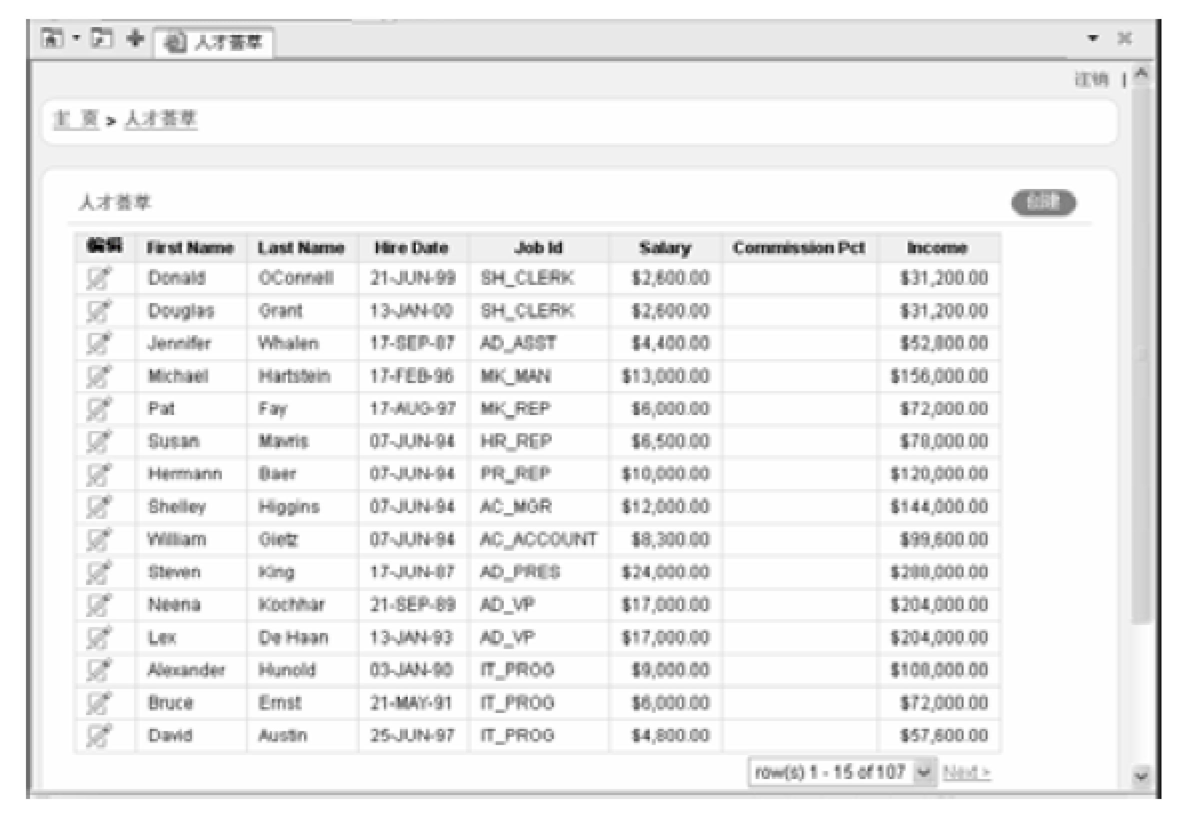


图　21.51　　　　图　21.52

接下来按照老板的要求，您将分别为 JOBS、EMPLOYEES 和 DEPARTMENTS 表创建值列表。

21.5　为JOBS创建值列表

首先您将为 JOBS 表创建一个值列表，具体操作步骤如下：

（1）在“主页”页面中单击最底部开发工具栏中的 Application 101 超链接，如图 21.53 所示。

（2）在“应用程序 101”页面单击“4-创建/编辑 人才信息”图标，如图 21.54 所示。

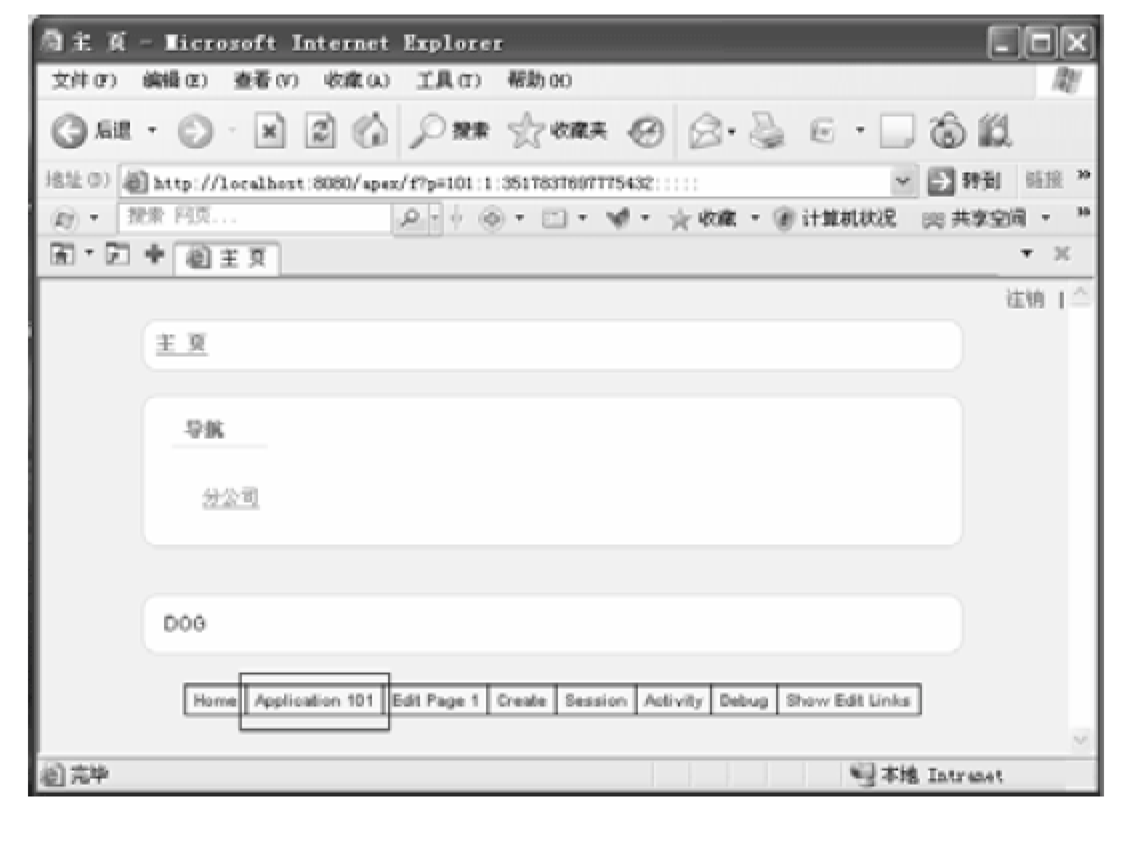


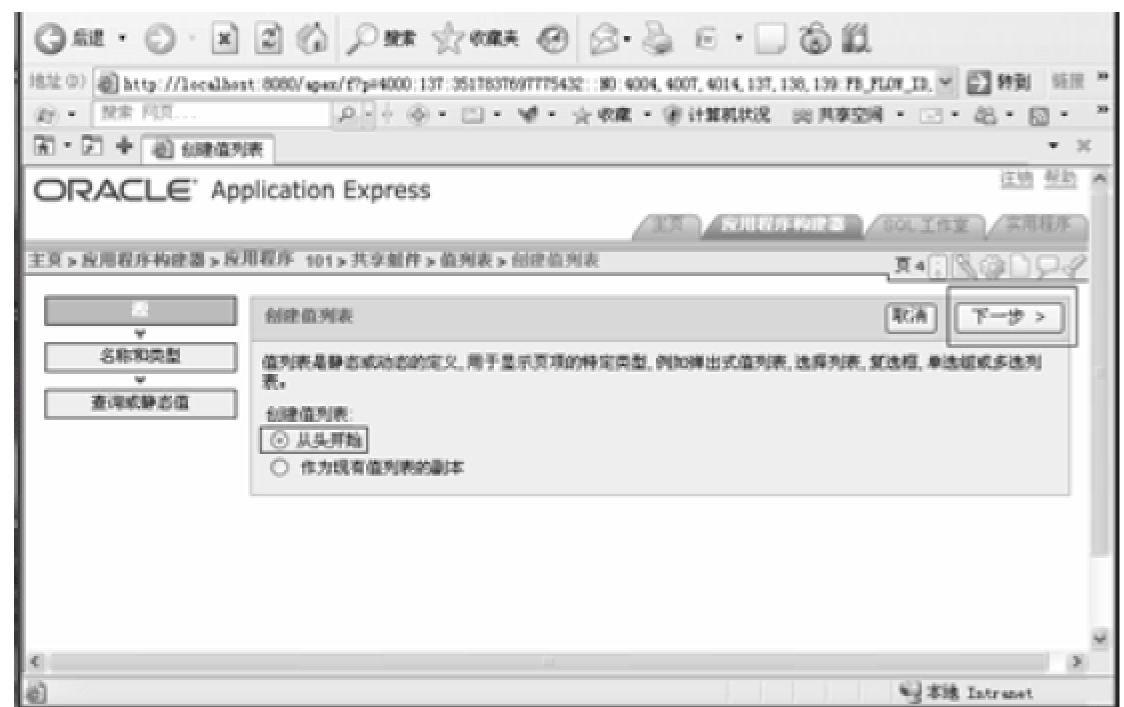


图　21.53　　　　图　21.54

（3）在“共享组件”区域的“值列表”栏中单击“创建”图标，如图 21.55 所示，将进入创建值列表向导界面。

（4）对于源，在“创建值列表”栏中接受默认选中的“从头开始”单选按钮并单击“下一步”按钮，如图 21.56 所示。


图 21.55


图 21.56

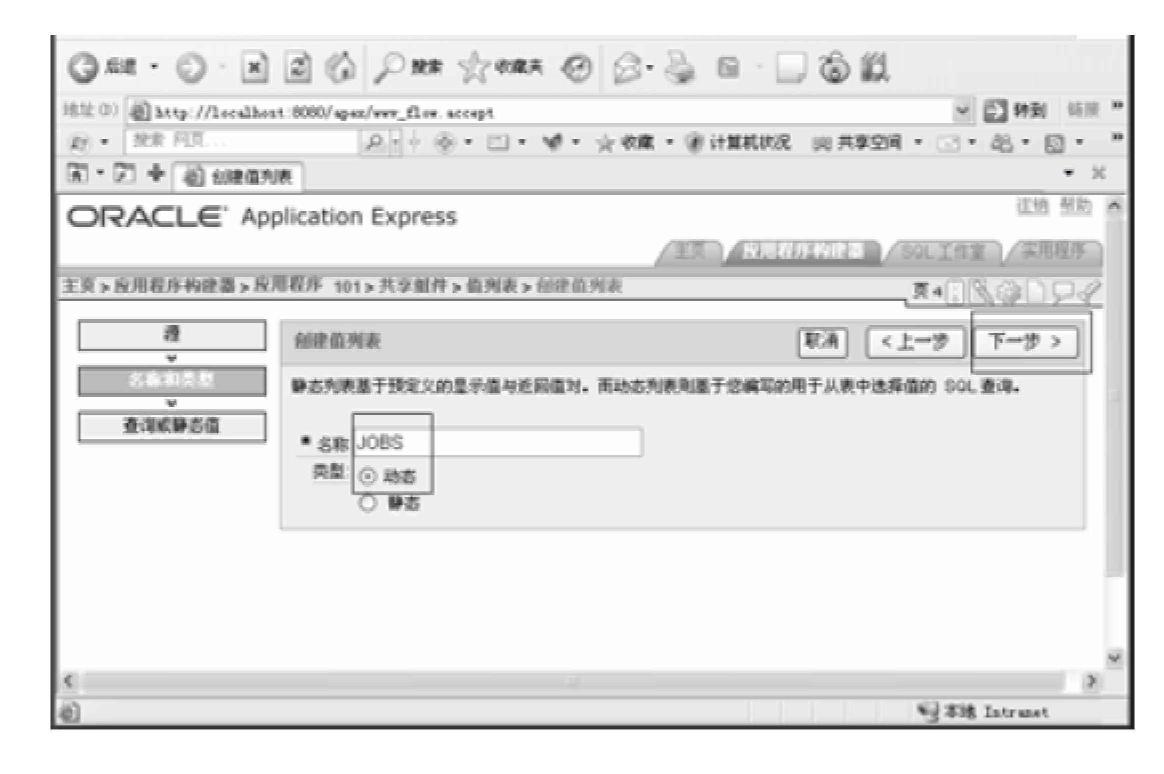
（5）在“名称”文本框中输入“JOBS”，在“类型”栏中选中“动态”单选按钮，单击“下一步”按钮，如图 21.57 所示。

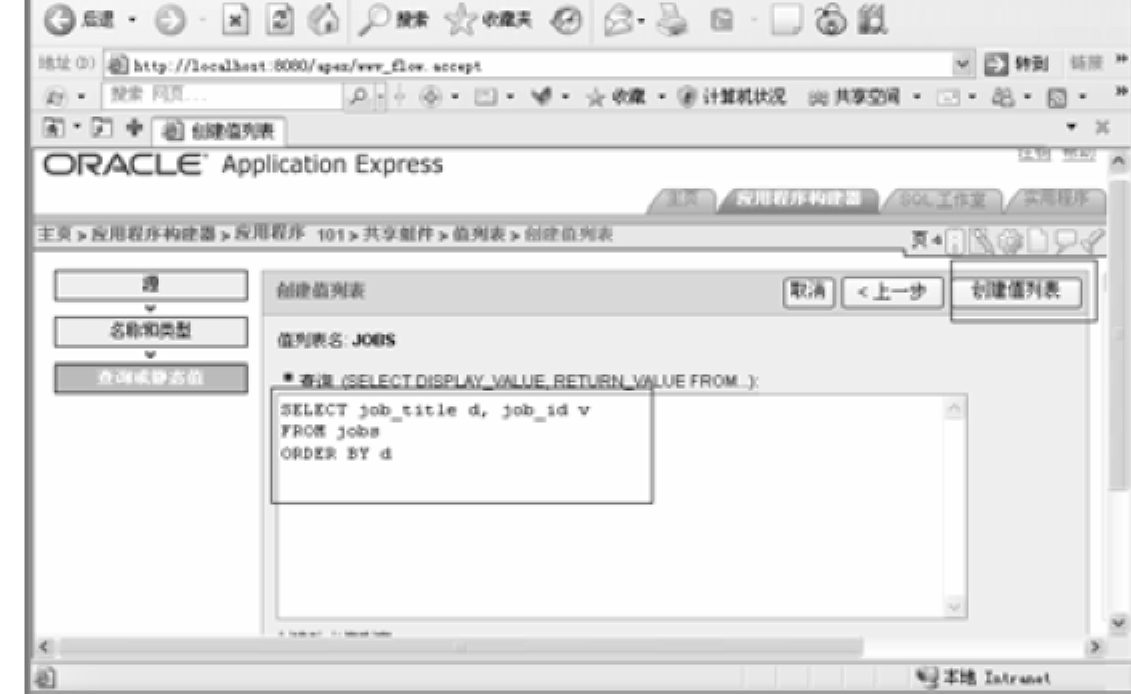
（6）在“查询”列表框中输入如下的 SQL 查询语句，用于替代已有的 SQL 语句（如图 21.58 所示）。

```
SELECT job_title d, job_id v
FROM jobs
ORDER BY d
```

查询的第 1 列显示的是给用户的值（d 是 display 的缩写），第 2 列是存入数据库或从数据库中取出的值（v 是 values 的缩写）。另外，随书的光盘上有一个 jobs.sql 的文件，该文件中的内容就是上面的 SQL 语句。如果读者没有学习过 Oracle SQL，可以将文件中的 SQL 语句复制到“查询”列表框中。

（7）单击“创建值列表”按钮，如图 21.58 所示。


图 21.57


图 21.58

之后就会出现如图 21.59 所示的页面。请读者注意，共享组件在它们被显式地加进该页之前是不会出现在该页的定义部分的。

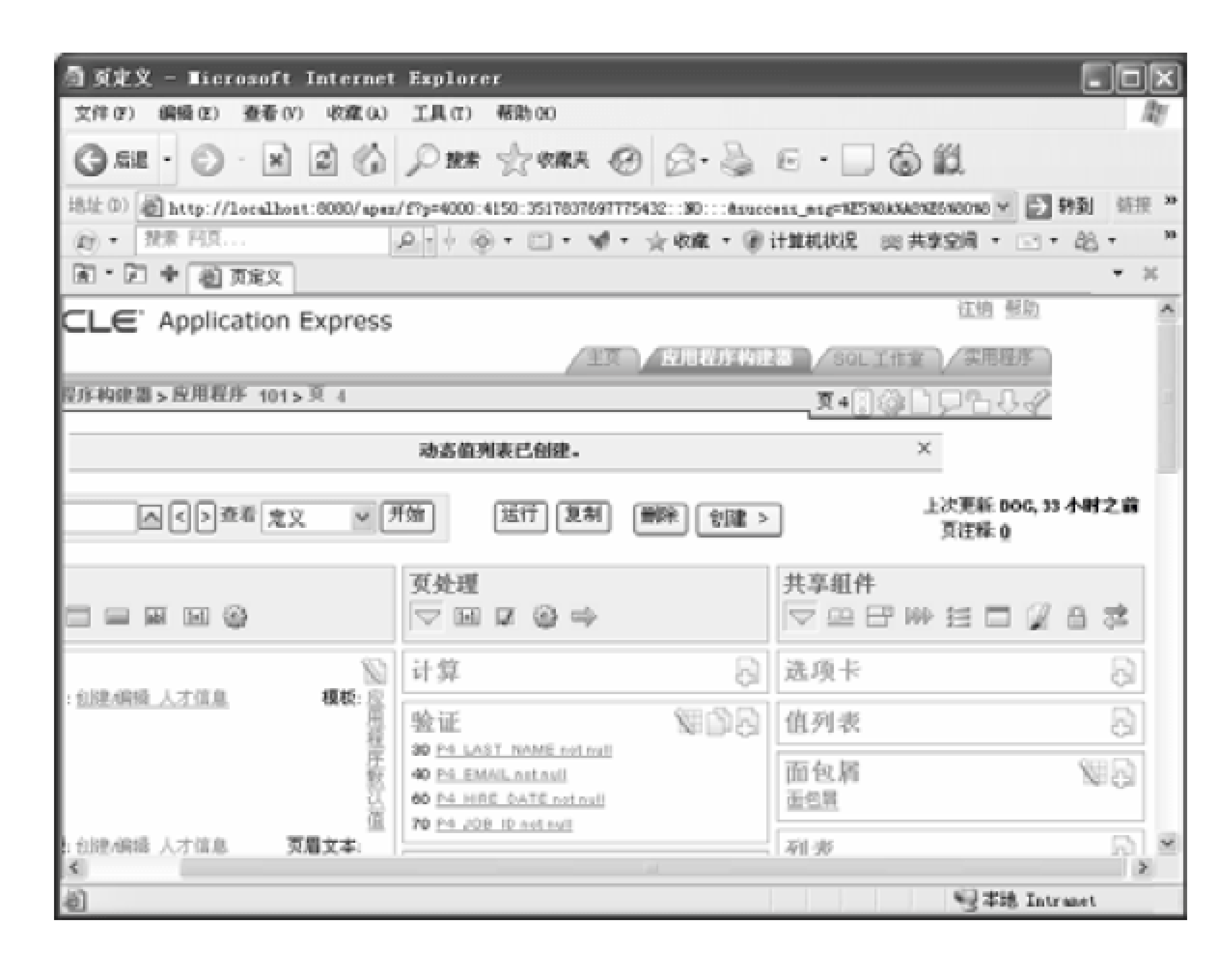


图　21.59

21.6　为EMPLOYEES创建值列表

下面您将为 EMPLOYEES 表创建一个值列表。以下是为 EMPLOYEES 表创建一个值列表的具体操作步骤：

（1）在“共享组件”区域的“值列表”栏中单击“创建”图标，如图 21.60 所示，将进入创建值列表向导界面。

（2）对于源，在“创建值列表”栏中接受默认选中的“从头开始”单选按钮并单击“下一步”按钮，如图 21.61 所示。

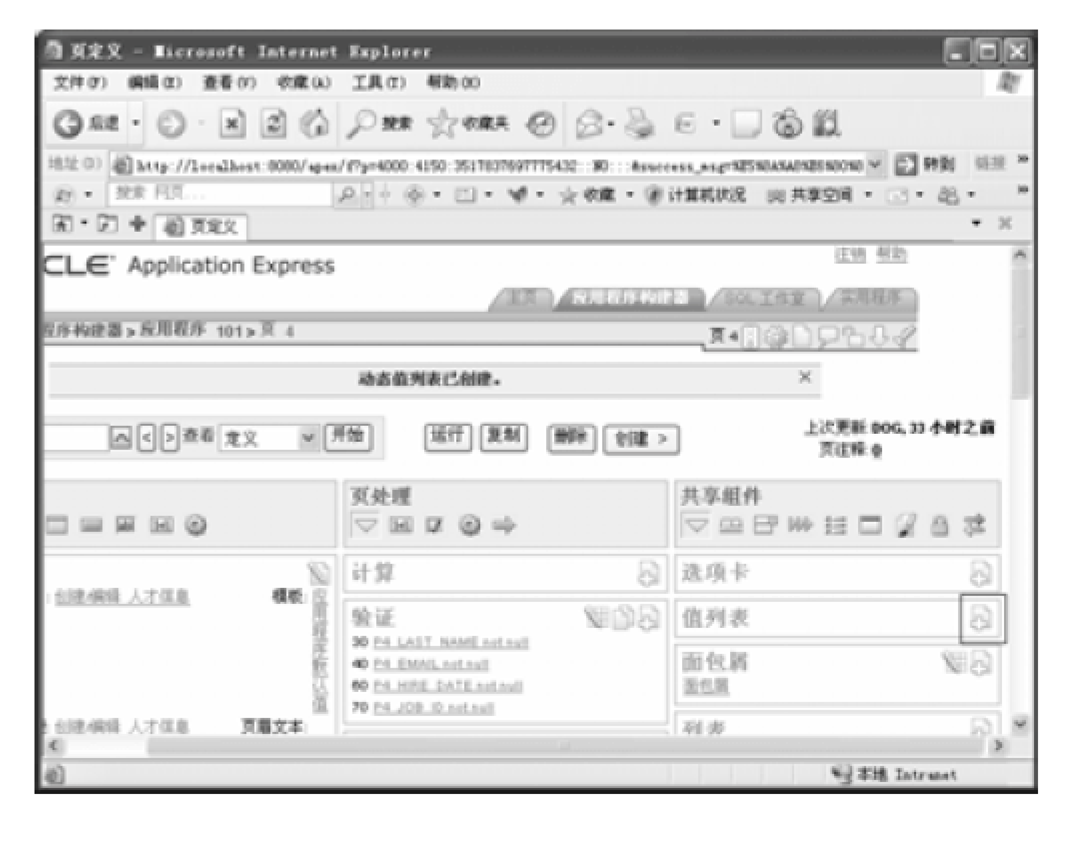


图　21.60

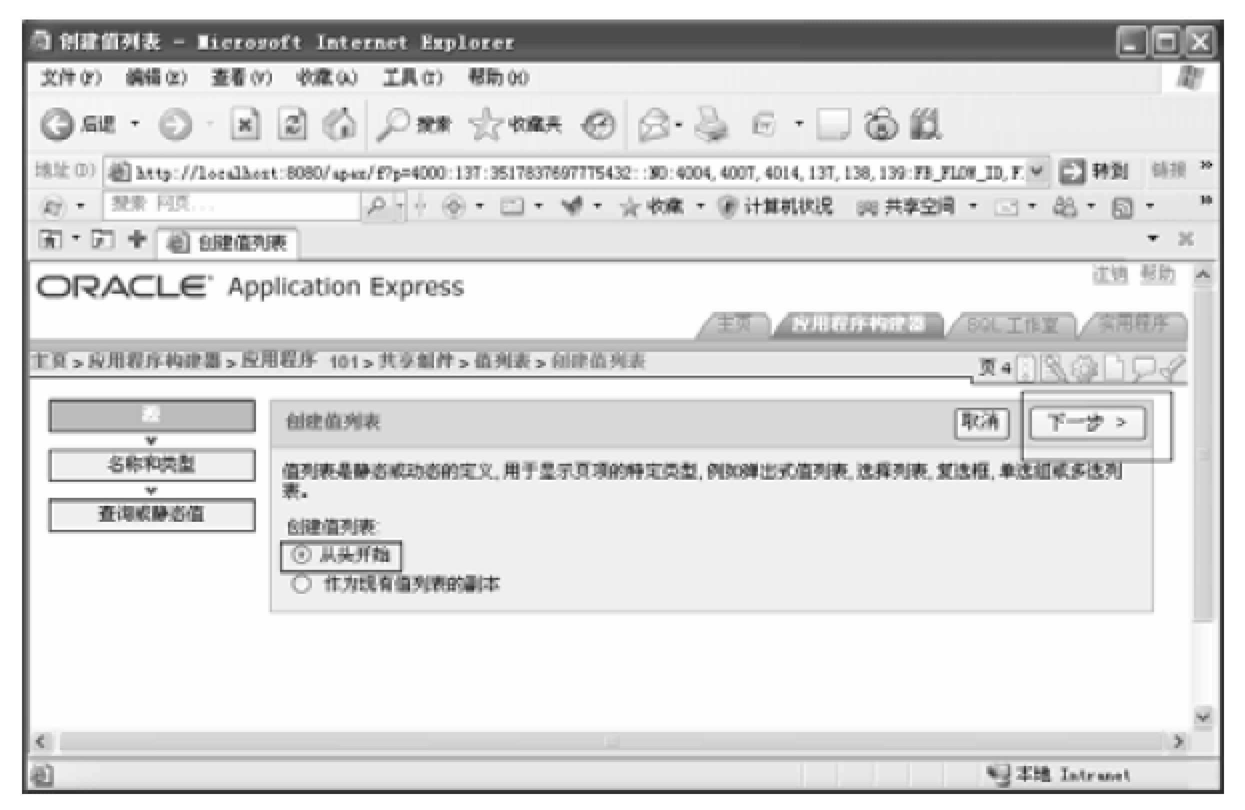


图　21.61

（3）在“名称”文本框中输入“EMPLOYEES”，在“类型”栏中选中“动态”单选按钮，然后单击“下一步”按钮，如图 21.62 所示。

（4）在“查询”列表框中输入如下的 SQL 查询语句，用于替代已有的 SQL 语句（如图 21.63 所示）。

```
SELECT first_name ||' '|| last_name d, employee_id v
FROM employees
ORDER BY last_name
```

另外，随书的光盘上有一个 employees.sql 文件，该文件中的内容就是上面的 SQL 语句。如果读者没有学习过 Oracle SQL，可以将文件中的 SQL 语句复制到“查询”列表框中。

（5）单击“创建值列表”按钮，如图 21.63 所示。

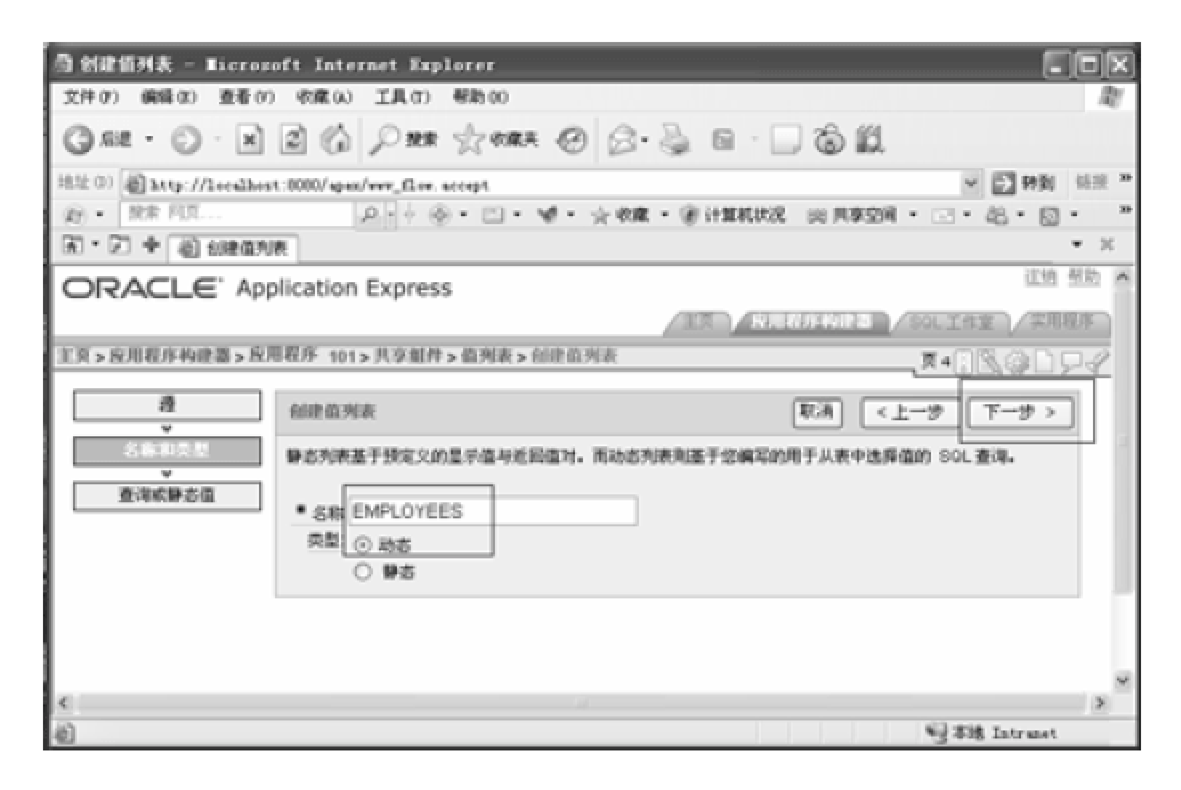


图　21.62



图　21.63

之后就会出现如图 21.64 所示的页面，这也就表示您已经成功地创建了一个基于 EMPLOYEES 表的值列表。



图　21.64

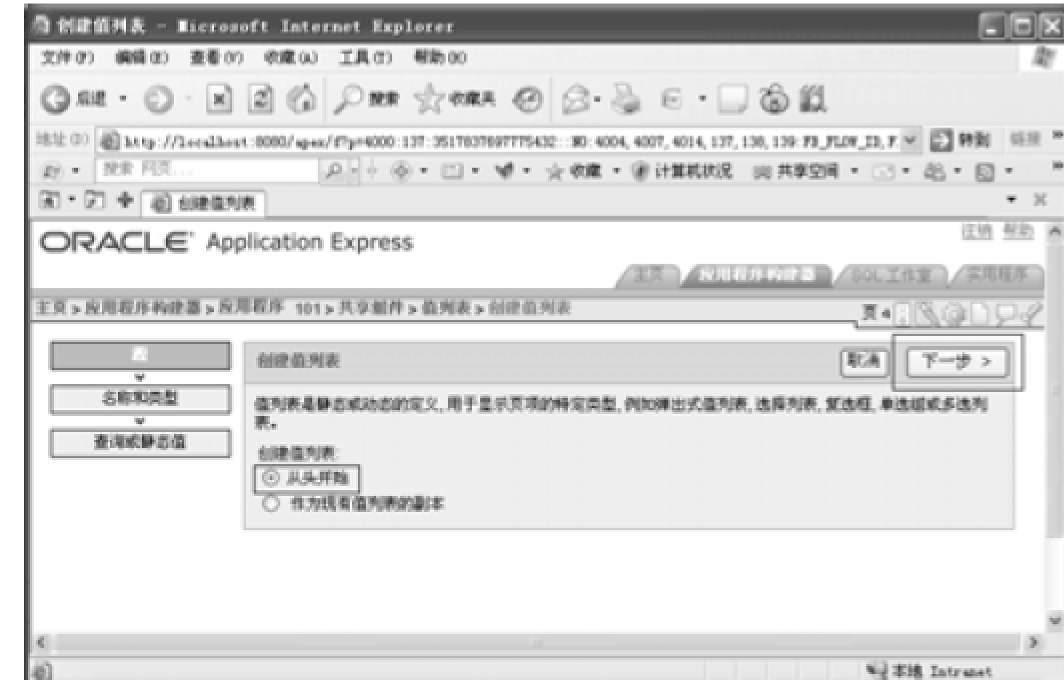
21.7　为DEPARTMENTS创建值列表

接下来您将为 DEPARTMENTS 表创建一个值列表，具体操作步骤如下：

（1）在“共享组件”区域的“值列表”栏中单击“创建”按钮，如图 21.65 所示，将进入创建值列表向导界面。

（2）对于源，在“创建值列表”栏中接受默认选中的“从头开始”单选按钮并单击“下一步”按钮，如图 21.66 所示。


图　21.65


图　21.66

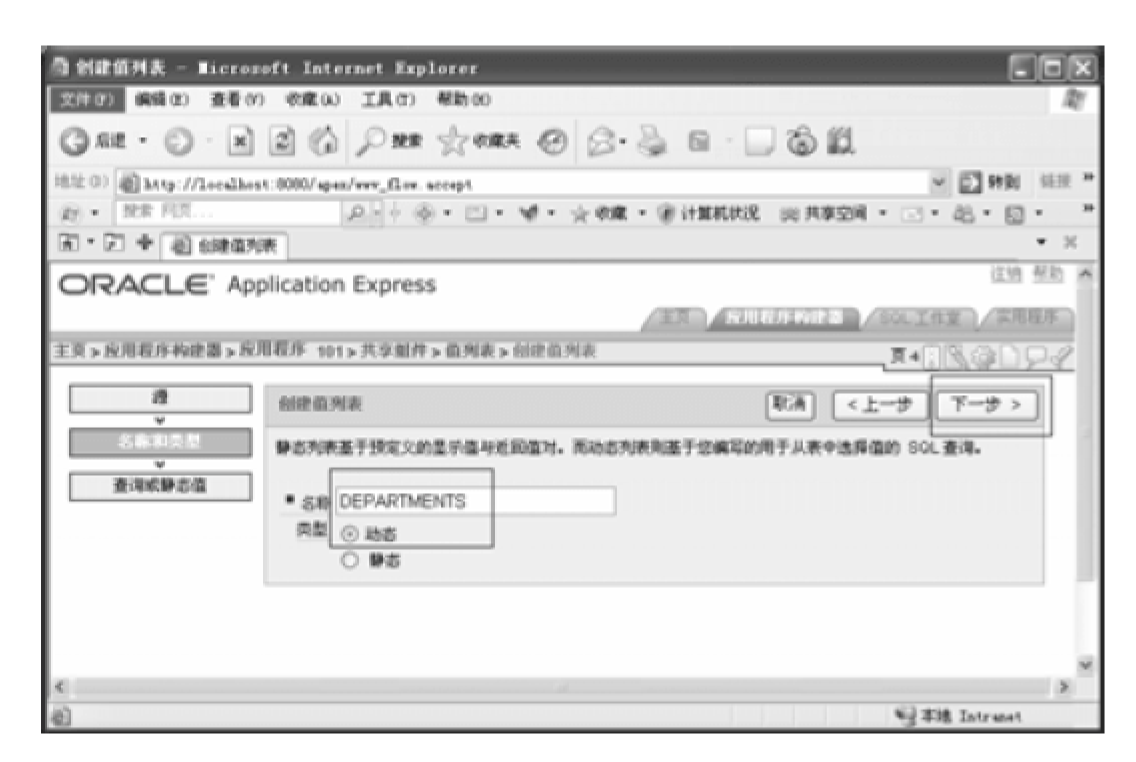
（3）在“名称”文本框中输入“DEPARTMENTS”，在“类型”栏中选中“动态”单选按钮，然后单击“下一步”按钮，如图 21.67 所示。

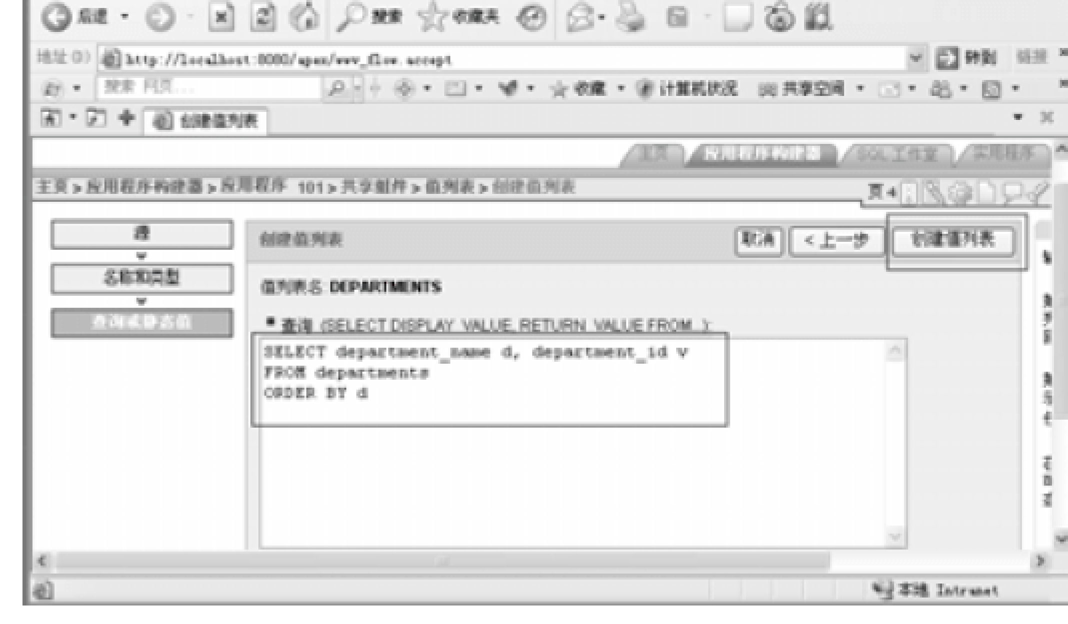
（4）在“查询”列表框中输入如下的 SQL 查询语句，用于替代已有的 SQL 语句（如图 21.68 所示）。

```
SELECT department_name d, department_id v
FROM departments
ORDER BY d
```

另外，随书的光盘上有一个 departments.sql 文件，该文件中的内容就是上面的 SQL 语句。如果读者没有学习过 Oracle SQL，可以将文件中的 SQL 语句复制到“查询”列表框中。

（5）单击“创建值列表”按钮，如图 21.68 所示。


图　21.67


图　21.68

之后就会出现如图 21.69 所示的页面，这也就表示您已经成功地创建了一个基于 DEPARTMENTS 表的值列表。

当创建了以上值列表之后，您需要将所对应的项改为以选择列表显示，因为在默认情况下这些项都是以正文字段来显示的。您还要将对应的项 P4_JOB_ID、P4_MANAGER_ID 和 P4_DEPARTMENT_ID 改名，这里的 P4 含义是页 4。



图 21.69

21.8 编辑JOB项

现在您首先编辑 P4_JOB_ID 项，具体的操作步骤如下：

（1）在页 4 的“页定义”页面使用最右面的滚动条向下滚动到“项”区域，如图 21.70 所示。

注意：

“项”区域列出了该页中所有的项。目前除了两个项以外，所有的项都是默认的正文字段。

（2）单击 P4_JOB_ID 超链接，如图 21.71 所示，将出现“编辑页项”页面。



图 21.70



图 21.71

（3）在“显示为”下拉列表框中选择“选择列表”选项，如图 21.72 所示。

（4）将“标签”改为“职位”，如图 21.73 所示。

（5）使用滚动条向下滚动到“值列表”区域，如图 21.74 所示。

（6）在“命名的 LOV”下拉列表框中选择 JOBS 选项，如图 21.75 所示。

（7）单击“应用更改”按钮，如图 21.76 所示。

之后将出现如图 21.77 所示的页面，您会发现 P4_JOB_ID 已经由“正文字段”变成了“选择列表”。



图　21.72

图　21.73



图　21.74



图　21.75



图　21.76



图　21.77

21.9　编辑MANAGER项

接下来您将编辑 P4_MANAGER_ID 项，具体的操作步骤如下：

（1）单击 P4_MANAGER_ID 超链接，如图 21.78 所示，将出现“编辑页项”页面。

（2）在“显示为”下拉列表框中选择“选择列表”选项，如图 21.79 所示。

（3）将“标签”改为“经理”，如图 21.80 所示。

（4）使用滚动条向下滚动到“值列表”区域，如图 21.81 所示。



图 21.78



图 21.79



图 21.80



图 21.81

（5）在“命名的 LOV”下拉列表框中选择 EMPLOYEES 选项，在“显示空值”下拉列表框中选择“是”选项，在“空显示值”文本框中输入“没有经理”，如图 21.82 所示。



图 21.82

提示：

在第 21.8 节编辑 JOB 项时，我们并未处理空值。这是因为 JOB_ID 在 JOBS 表中是不能为空的（以 HR 用户启动 SQL*Plus 之后使用 desc jobs 就可以得到这一信息）。但是 MANAGER_ID 在 EMPLOYEES 中是可选项，即可以为空（使用 desc EMPLOYEES 命令可以得到这一信息），所以就需要处理空值。

（6）单击“应用更改”按钮，如图 21.83 所示。

之后将出现如图 21.84 所示的页面，您会发现 P4_MANAGER_ID 已经由“正文字段”

变成了“选择列表”。



图　21.83



图　21.84

21.10　编辑DEPARTMENT项

最后您将编辑 P4_DEPARTMENT_ID 项，具体的操作步骤如下：

（1）单击 P4_DEPARTMENT_ID 超链接，如图 21.85 所示，将出现“编辑页项”页面。

（2）在“显示为”下拉列表框中选择“选择列表”选项，如图 21.86 所示。



图　21.85



图　21.86

（3）将“标签”改为“分公司”，如图 21.87 所示。

（4）使用滚动条向下滚动到“值列表”区域，如图 21.88 所示。

（5）在“命名的 LOV”下拉列表框中选择 DEPARTMENTS 选项，在“显示空值”下拉列表框中选择“是”选项，在“空显示值”文本框中输入“未分配分公司/部门”，如图 21.89 所示。

提示：

因为 DEPARTMENT_ID 在 EMPLOYEES 中也是可选项，即可以为空（使用 desc EMPLOYEES 命令可以得到这一信息），所以也需要处理空值。

（6）单击“应用更改”按钮，如图 21.90 所示。



图 21.87



图 21.88



图 21.89



图 21.90

之后将出现如图 21.91 所示的页面，您会发现 P4_DEPARTMENT_ID 已经由“正文字段”变成了“选择列表”。



图 21.91

到此为止，您已经成功地编辑了所有的所需项，之后就要对它们进行预览和测试了。

21.11 运行并预览网页

下面就是运行并预览相关网页的具体操作步骤：

（1）在“页 4”页面中单击“运行页”按钮，如图 21.92 所示。

之后出现如图 21.93 所示的页面。正像您所见到的，在“职位”、“经理”和“分公

司”的最右面都出现了一个下拉按钮。



图　21.92



图　21.93

（2）单击“职位”右面的下拉按钮，您将获取所有的职位，如图 21.94 所示。

（3）单击“经理”右面的下拉按钮，您将获取所有的经理，如图 21.95 所示。



图　21.94

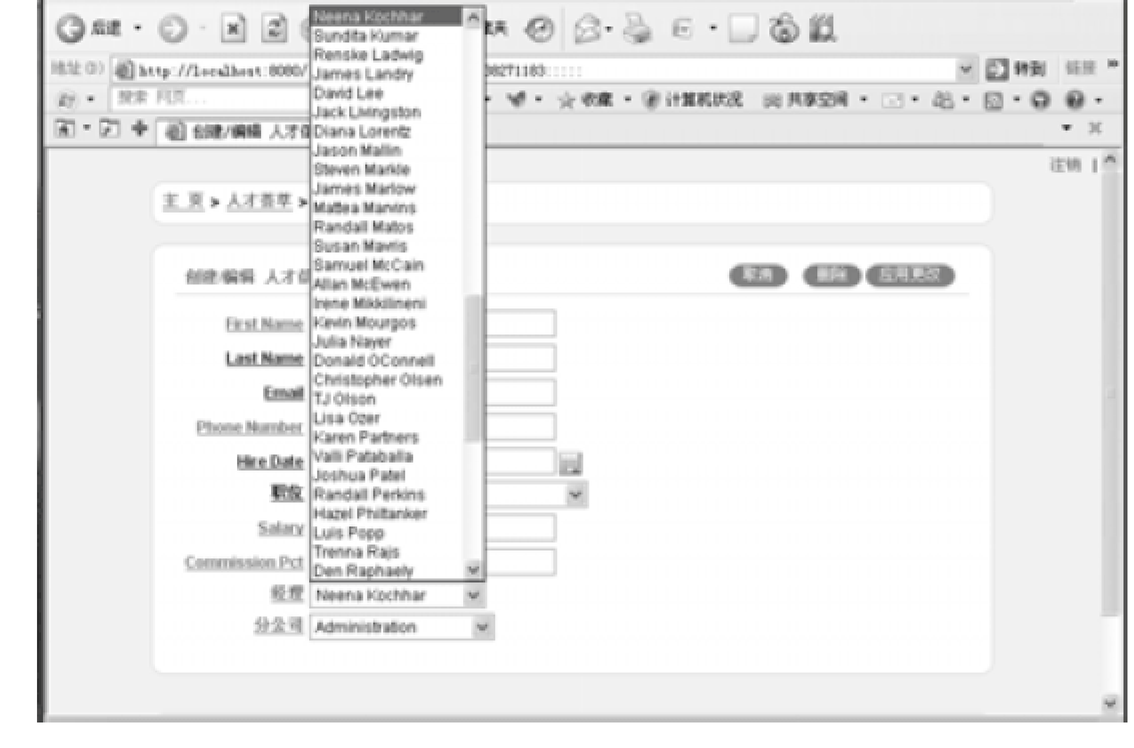


图　21.95

（4）单击“分公司”右面的下拉按钮，您将获取所有的分公司（部门），如图 21.96 所示。

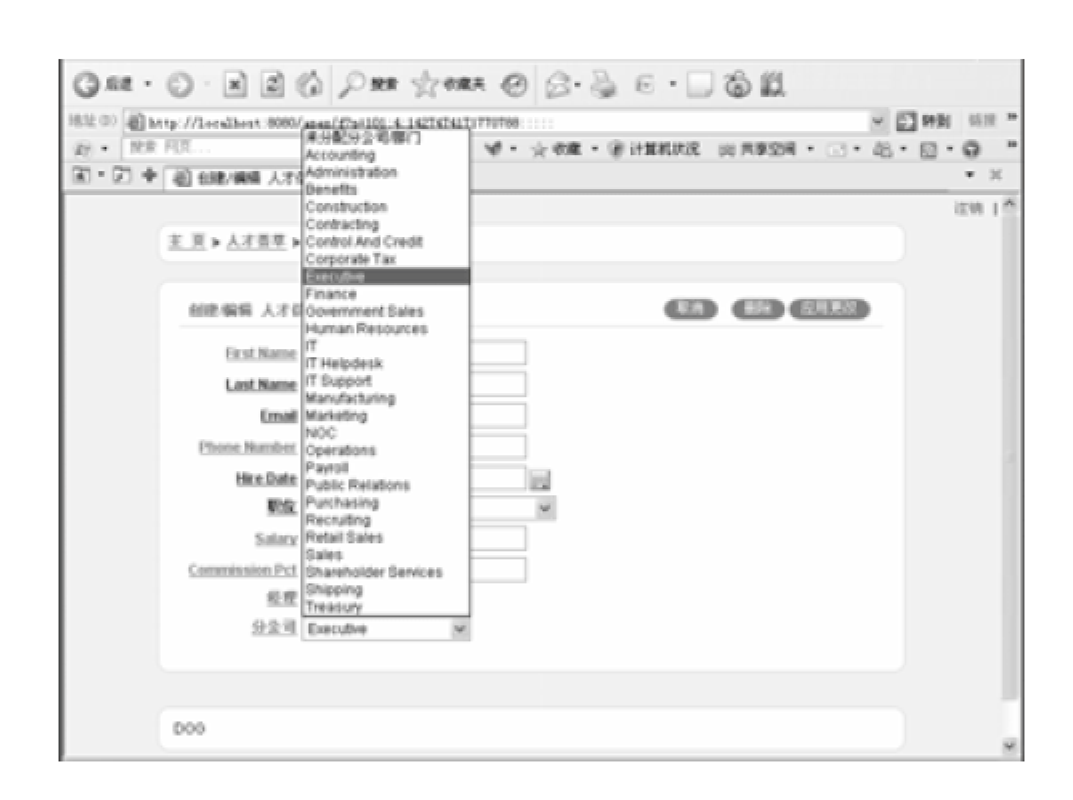


图　21.96

接下来您最好测试一下您所定义的空值部分是否正常工作，具体步骤如下：

（1）在 DOS 窗口中使用例 21-1 的命令以 HR 用户登录数据库。

例 21-1

```
sqlplus hr/hr
```

（2）使用例 21-2 的 SQL 查询语句获得没有经理的员工的信息。

例 21-2

```
SQL> select first_name, last_name
  2  from employees
  3  where manager_id is null;
```

例 21-2 结果

```
FIRST_NAME           LAST_NAME
-------------------- -------------
Steven               King
```

（3）使用例 21-3 的 SQL 查询语句获得不属于任何部门的员工的信息。

例 21-3

```
SQL> select first_name, last_name
  2  from employees
  3  where department_id is null;
```

例 21-3 结果

```
FIRST_NAME           LAST_NAME
-------------------- ----------
Kimberely             Grant
```

（4）在面包屑处单击“人才荟萃”超链接，如图 21.97 所示，然后将退到“人才荟萃”页面。

（5）单击没有经理的名为 Steven King 的员工记录最左面的编辑按钮，如图 21.98 所示。



图 21.97

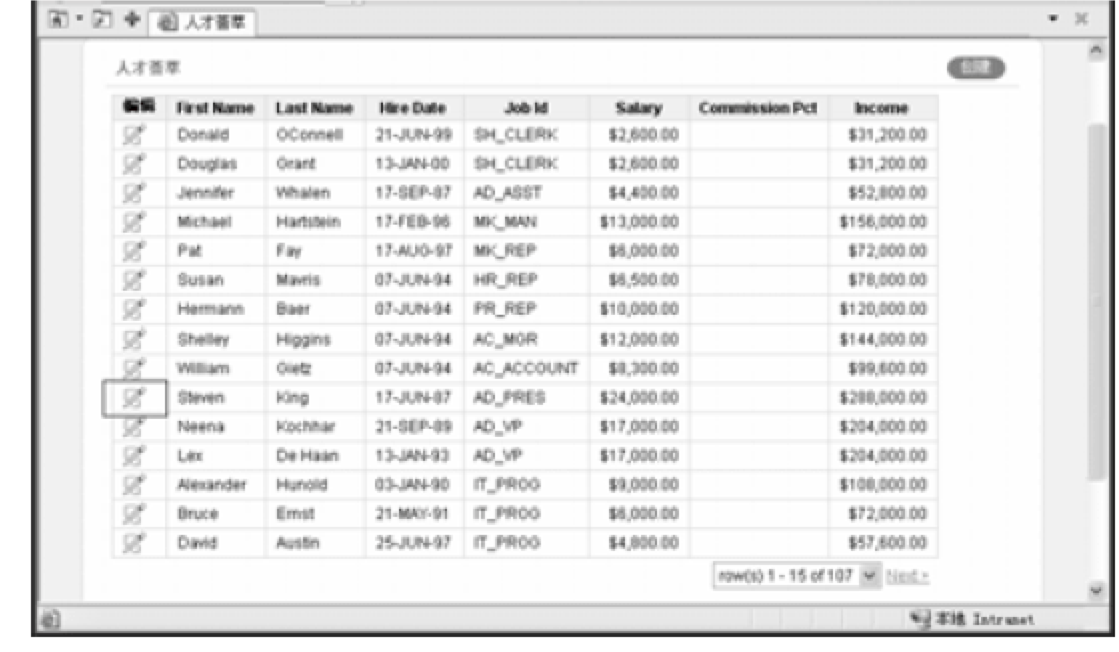


图 21.98

之后就出现了如图 21.99 所示的页面，果然在“经理”下拉列表框中显示的为“没有经理”，之后再退回到“人才荟萃”页面。



图　21.99

（6）单击不属于任何部门的名为 Kimberely Grant 的员工记录最左面的编辑按钮，如图 21.100 所示。

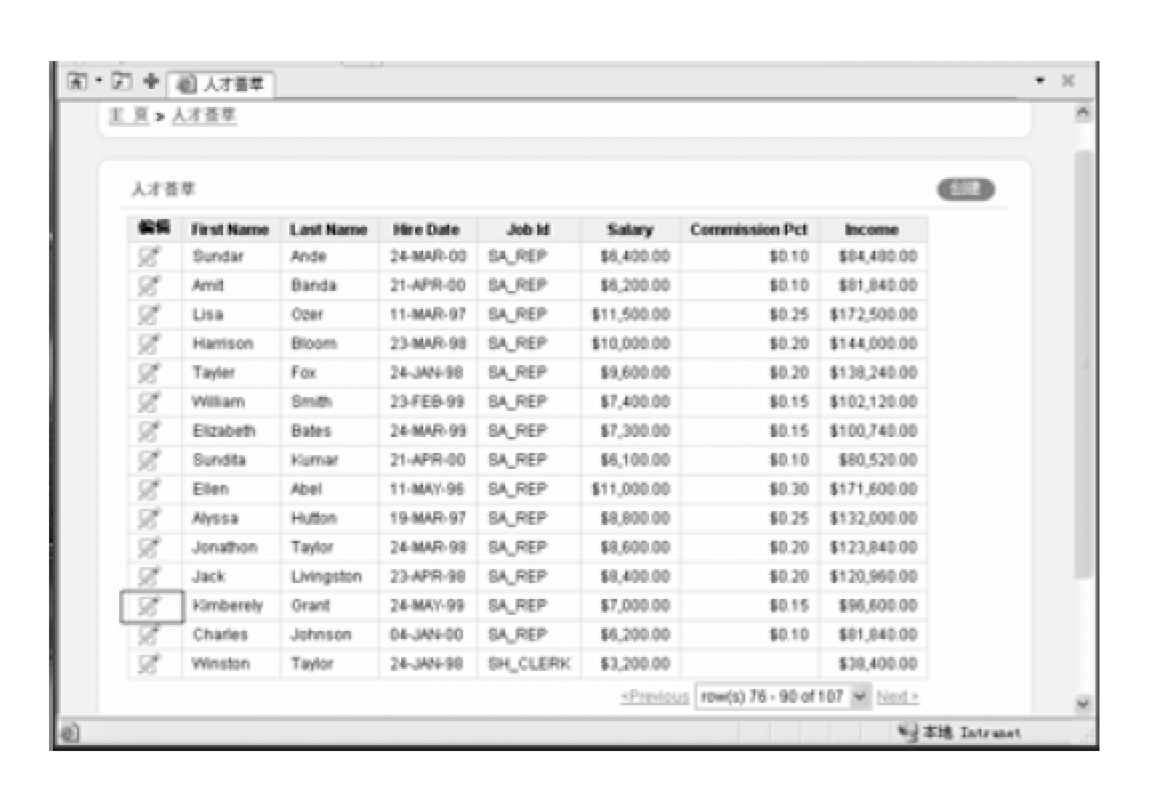


图　21.100

之后就出现了如图 21.101 所示的页面，果然在“分公司”下拉列表框中显示的为“未分配分公司/部门”。

经过以上的测试您可以放心了，因为您已经按照老板的要求，成功地分别为 JOBS、EMPLOYEES 和 DEPARTMENTS 创建了值列表。但是一想到老板对比她的驴肉烧饼还古老的汉字的着迷程度，您心里就又开始紧张起来。



图　21.101

21.12　汉化报表的显示

既然通过在 SQL 查询语句中为列定义别名的方法行不通，您就得另辟蹊径。所谓“通往十三陵的路不止一条”，可以通过修改列标题的方法来将报表列标题的显示汉化，利用这种方法汉化可以继续保持报表的编辑功能，具体操作步骤如下：

（1）在“应用程序 101”页面中单击“3-人才荟萃”图标，如图 21.102 所示。之后将出现页 3 的“页定义”页面。

（2）在“区域”栏中单击“报表”超链接，如图 21.103 所示。之后将出现“报表属性”页面。



图　21.102



图　21.103

（3）将 FIRST_NAME 的标题改为“名”，将 LAST_NAME 的标题改为“姓”，将 HIRE_DATE 的标题改为“雇用日期”，将 JOB_ID 的标题改为“职位号”，将 SALARY 的标题改为“工资”，将 COMMISSION_PCT 的标题改为“佣金比率”，将 INCOME 的标题改为“年收入”，如图 21.104 所示。

（4）单击“应用更改”按钮来确认所做的修改，如图 21.105 所示。之后又将回到页 3 的“页定义”页面。



图　21.104



图　21.105

（5）单击“运行页”按钮，如图 21.106 所示。之后可能会出现登录界面。



图　21.106

（6）如果出现登录界面，在“用户名”文本框中输入“dog”，在“口令”文本框中输入 xm_Q1ng，最后单击“登录”按钮，如图 21.107 所示。

最后就会出现如图 21.108 所示的“人才荟萃”详细信息，可以看出所有的列标题都已经是汉字了，同时每一行上的编辑按钮依然存在。

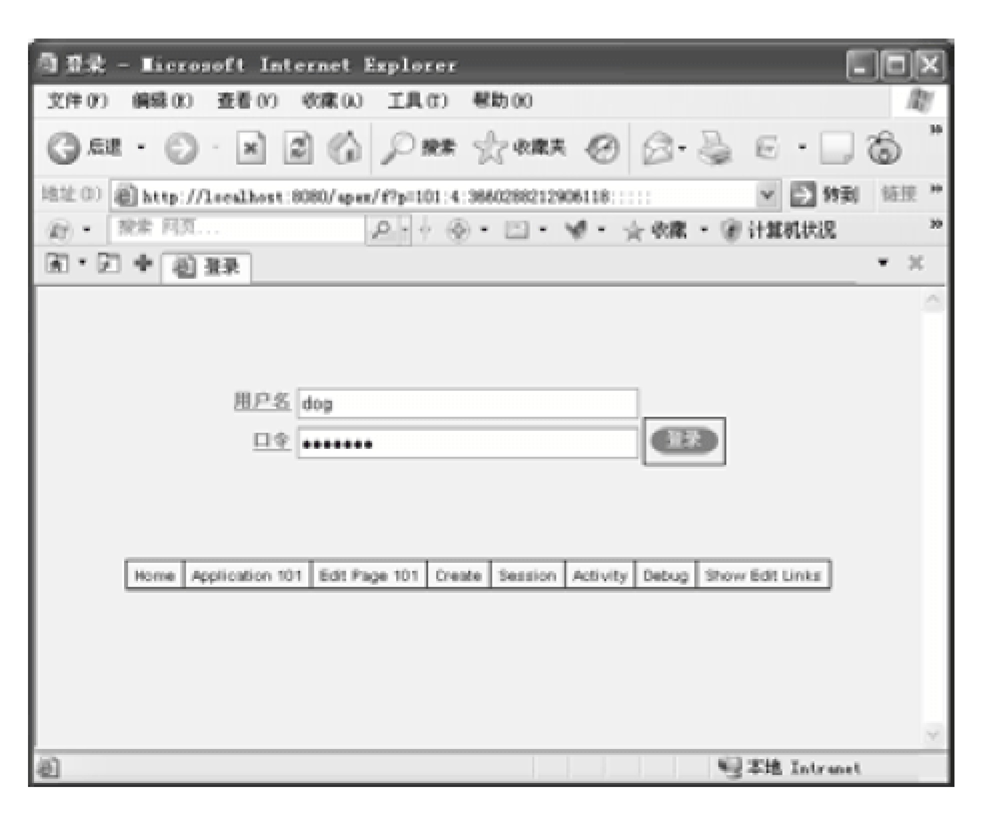


图　21.107

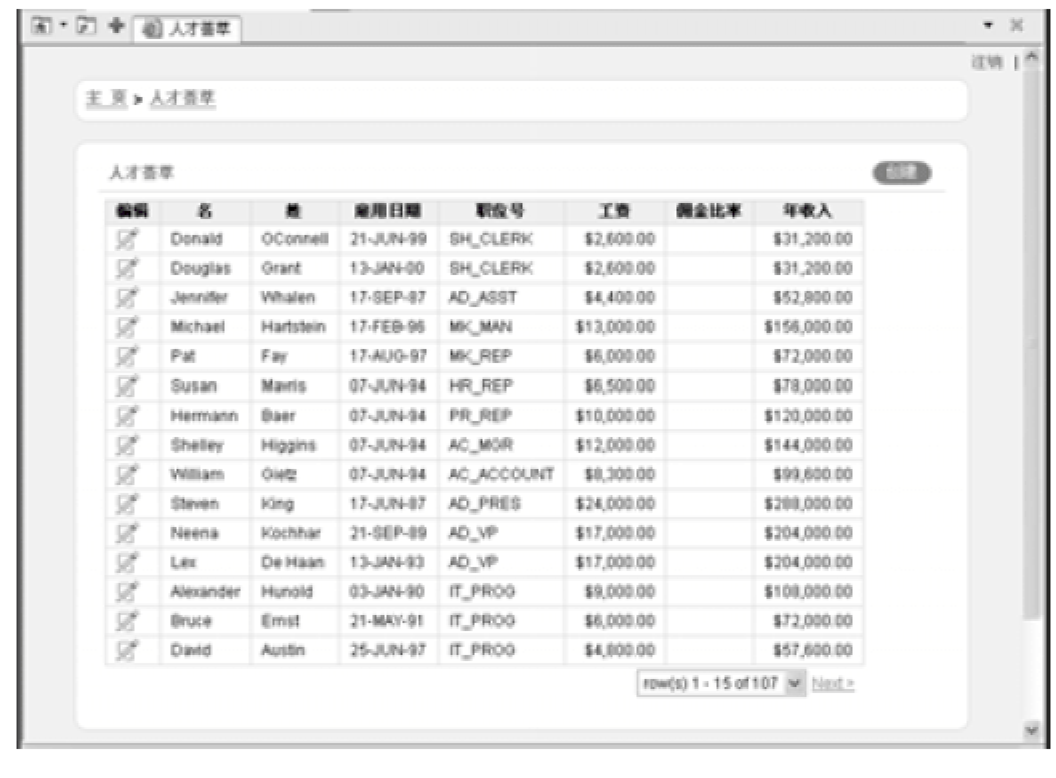


图　21.108

21.13　汉化表单的显示

与报表列标题的显示汉化不同，可以通过修改标签的方法来将列表字段标题的显示汉化，具体步骤如下：

（1）在“应用程序 101”页面中单击“4-创建/编辑 人才信息”图标，如图 21.109 所示。之后将出现页 4 的“页定义”页面。

（2）在“项”区域单击 P4_FIRST_NAME 超链接，如图 21.110 所示。之后将出现页项 P4_FIRST_NAME 的“编辑页项”界面。

（3）在“标签”区域将原有的标签改为“名”后，单击“应用更改”按钮，如图 21.111 所示。

（4）重复类似第（3）步的操作，将 P4_LAST_NAME 的标签改为“姓”，P4_EMAIL 的标签改为“电子邮件”，P4_PHONE_NUMBER 的标签改为“电话号码”，P4_HIRE_DATE

的标签改为“雇用日期”、P4_SALARY_ID 的标签改为“工资”，P4_COMMISSION_PCT 的标签改为“佣金比率”。

（5）单击“运行页”按钮来运行“页 4”，如图 21.112 所示。

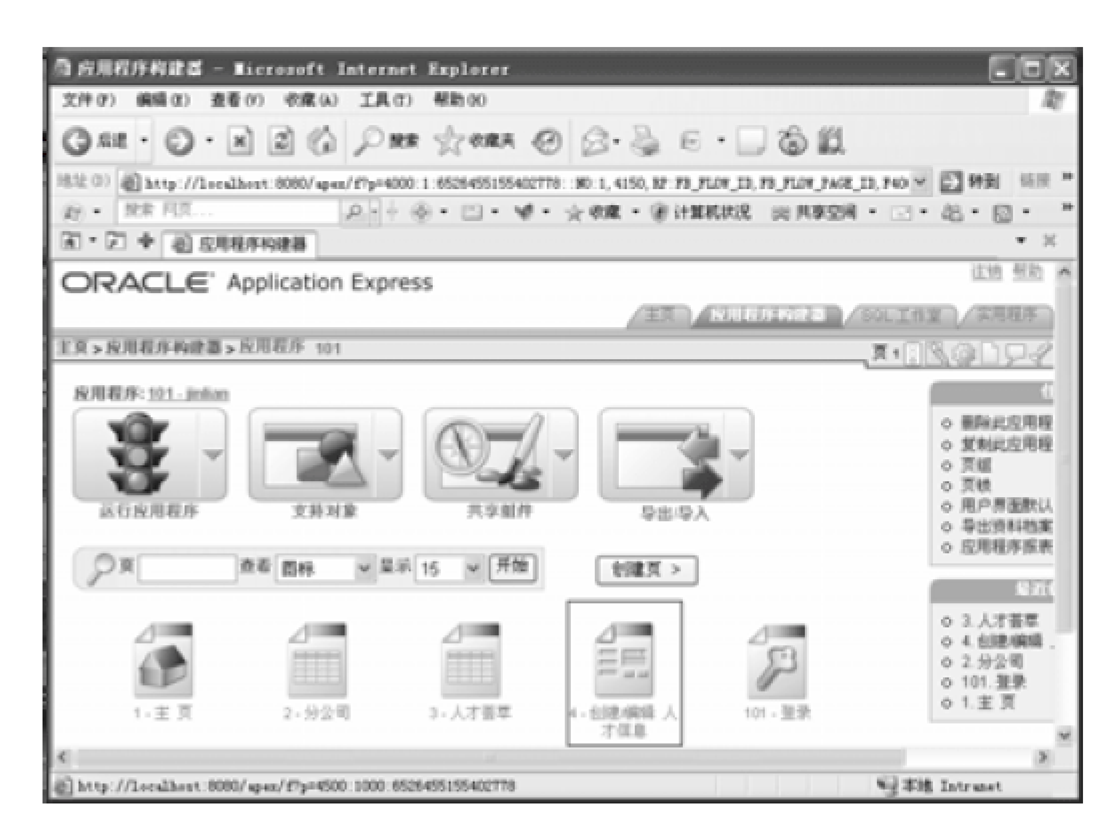


图 21.109



图 21.110



图 21.111



图 21.112

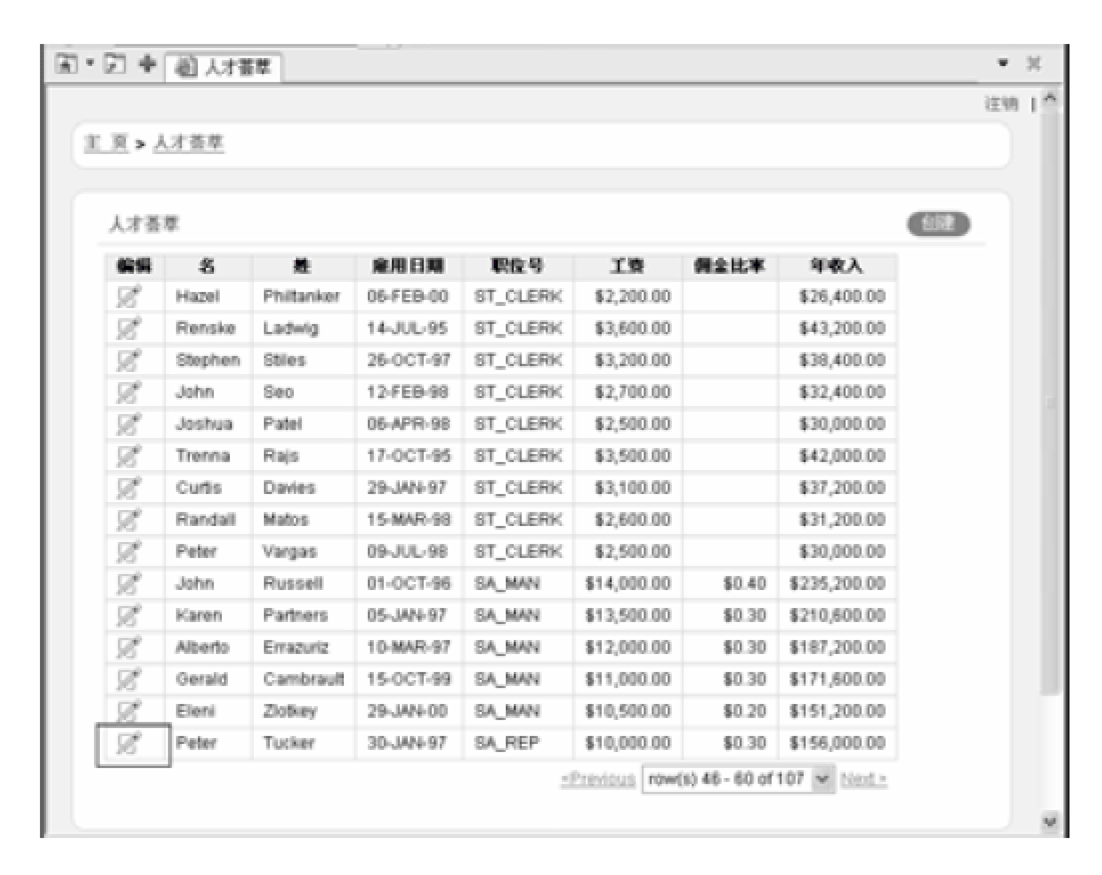
（6）在面包屑中单击“人才荟萃”超链接以退回到“人才荟萃”页面，如图 21.113 所示。



图 21.113

（7）单击某一行的“编辑”按钮以转到与该行记录相对的表单页面，如图 21.114 所示。

最后将出现“创建/编辑 人才信息”表单页面，如图 21.115 所示。表单中所有字段的名称都是汉字，这表明您的工作已经可以暂告一个段落了。



编辑	名	姓	雇用日期	职位号	工资	佣金比率	年收入
	Hazel	Philtanker	06-FEB-00	ST_CLERK	$2,200.00		$26,400.00
	Renske	Ladwig	14-JUL-95	ST_CLERK	$3,600.00		$43,200.00
	Stephen	Stiles	26-OCT-97	ST_CLERK	$3,200.00		$38,400.00
	John	Seo	12-FEB-98	ST_CLERK	$2,700.00		$32,400.00
	Joshua	Patel	06-APR-98	ST_CLERK	$2,500.00		$30,000.00
	Trenna	Rajs	17-OCT-95	ST_CLERK	$3,500.00		$42,000.00
	Curtis	Davies	29-JAN-97	ST_CLERK	$3,100.00		$37,200.00
	Randall	Matos	15-MAR-98	ST_CLERK	$2,600.00		$31,200.00
	Peter	Vargas	09-JUL-98	ST_CLERK	$2,500.00		$30,000.00
	John	Russell	01-OCT-96	SA_MAN	$14,000.00	$0.40	$235,200.00
	Karen	Partners	05-JAN-97	SA_MAN	$13,500.00	$0.30	$210,600.00
	Alberto	Errazuriz	10-MAR-97	SA_MAN	$12,000.00	$0.30	$187,200.00
	Gerald	Cambrault	15-OCT-99	SA_MAN	$11,000.00	$0.30	$171,600.00
	Eleni	Zlotkey	29-JAN-00	SA_MAN	$10,500.00	$0.20	$151,200.00
	Peter	Tucker	30-JAN-97	SA_REP	$10,000.00	$0.30	$156,000.00



图　21.114　　　　图　21.115

当您将所开发的网页演示给老板之后，她老人家自然高兴万分。令她惊喜的是，您不但这么快地开发出了对公司来说具有里程碑意义的互联网应用程序，而且所有的标题显示都是看上去同她的驴肉烧饼一样优美的汉字。

第22章 在网页中加入链接

尽管您已经在 jinlian-101 应用程序中成功地创建了“3-人才荟萃”和“4-创建/编辑 人才信息”网页，但是在该应用程序的主页却看不到它们，如图 22.1 所示。这样操作起来很不方便，看上去也不专业，因此您将把“3-人才荟萃”的超链接添加到主页中。



图 22.1

22.1 在主页上添加“人才荟萃”报表的超链接

本节您将在主页上添加一个转到“人才荟萃”（员工）报表的超链接，这个超链接将出现在主页左侧的导航区域，以下就是操作的具体步骤：

（1）在“主页”的编辑页面上单击“应用程序构建器”图标，如图 22.2 所示。

（2）在“应用程序构建器”的编辑页面上单击 jinlian-101 图标，如图 22.3 所示。

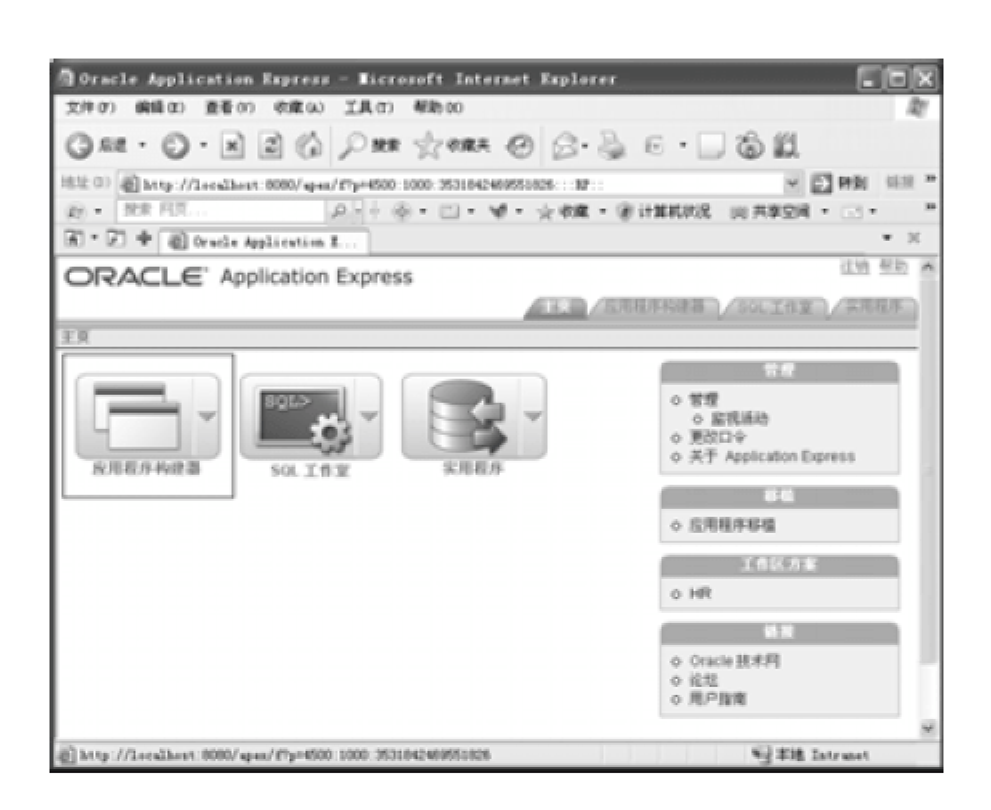


图 22.2

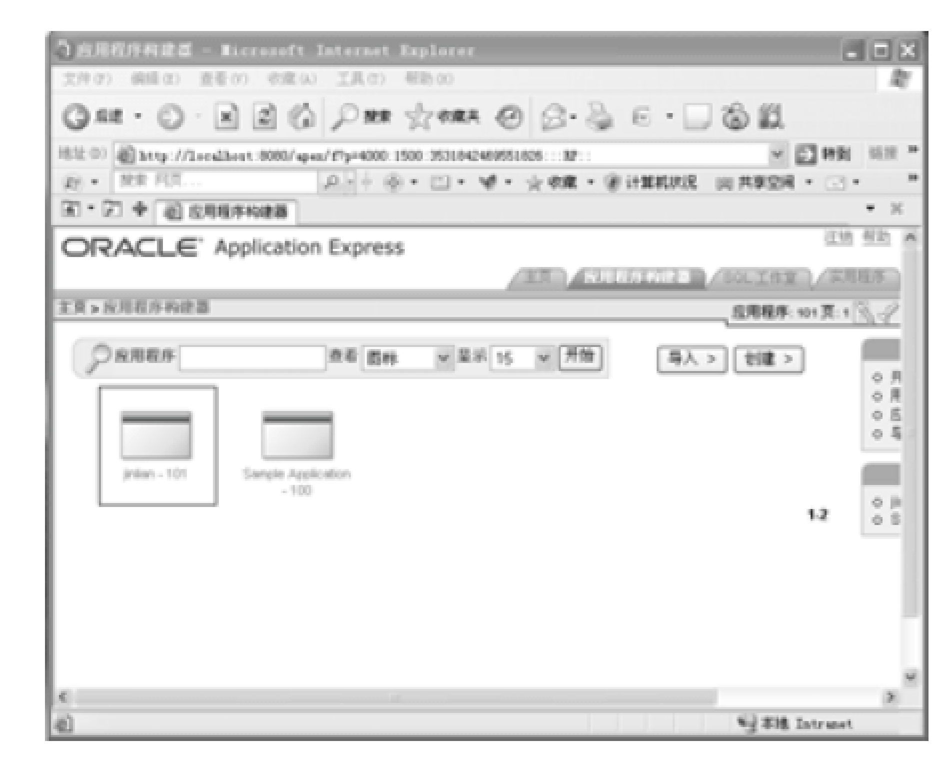


图 22.3

（3）在“应用程序 101”的编辑页面上单击“主页”图标，如图 22.4 所示。

（4）在页 1 的编辑页面的“区域”栏中单击“列表”超链接，如图 22.5 所示。

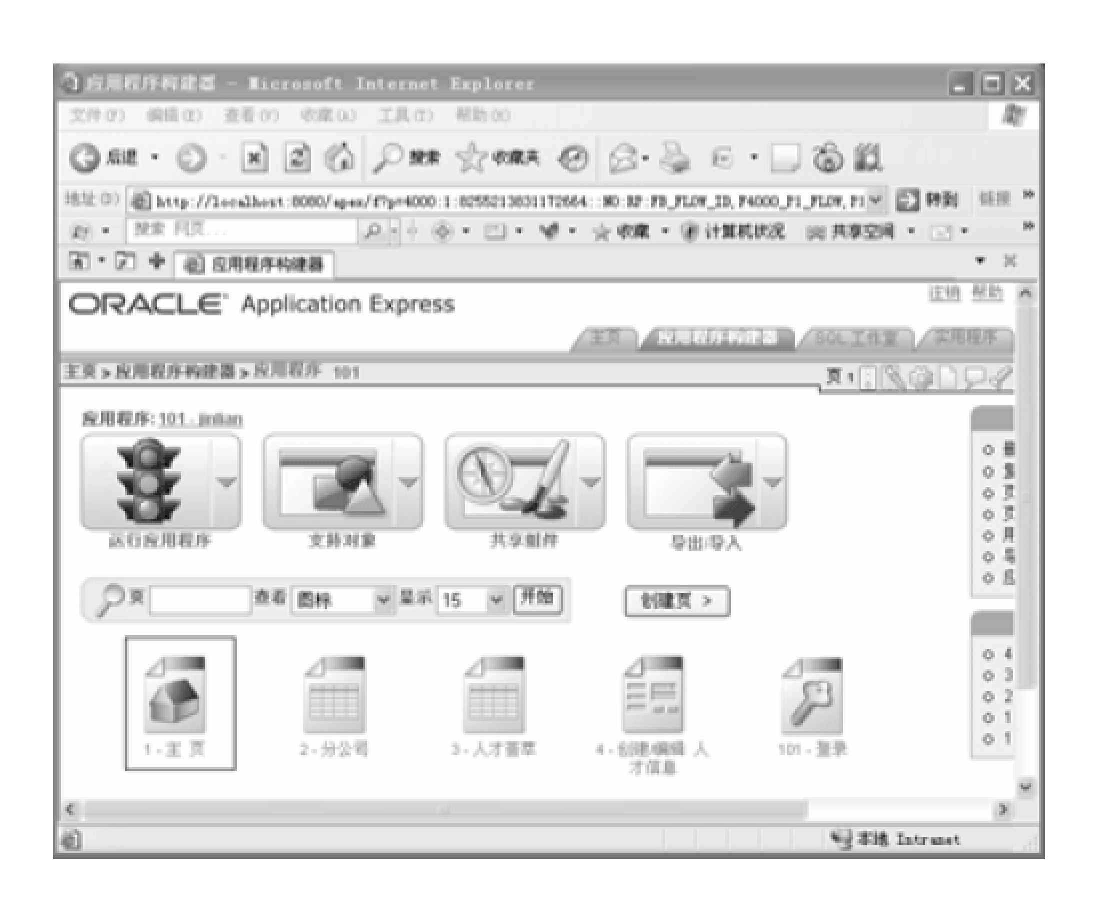


图　22.4



图　22.5

（5）在“列表条目”的编辑页面中单击右侧的“创建列表条目”按钮，如图 22.6 所示。

（6）在“创建/编辑列表条目”页面上将“序列”修改为 20，在“列表条目标签”列表框中输入“人才荟萃”，在“页”文本框中输入“3”，最后单击“创建”按钮，如图 22.7 所示。



图　22.6



图　22.7

（7）在“列表条目”的编辑页面上单击该页右上角的“运行”按钮，如图 22.8 所示。

（8）打开登录界面，在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”，如图 22.9 所示。

（9）单击“登录”按钮，如图 22.9 所示。之后将出现 jinlian 101 应用程序的主页，此时在主页中已经包含了一个“人才荟萃”超链接。

（10）单击“人才荟萃”超链接进行测试，如图 22.10 所示。

之后将出现“人才荟萃”（员工）报表，如图 22.11 所示。这表明您已经成功地在主页上添加了一个将用户转到“人才荟萃”（员工）报表的超链接。



图 22.8

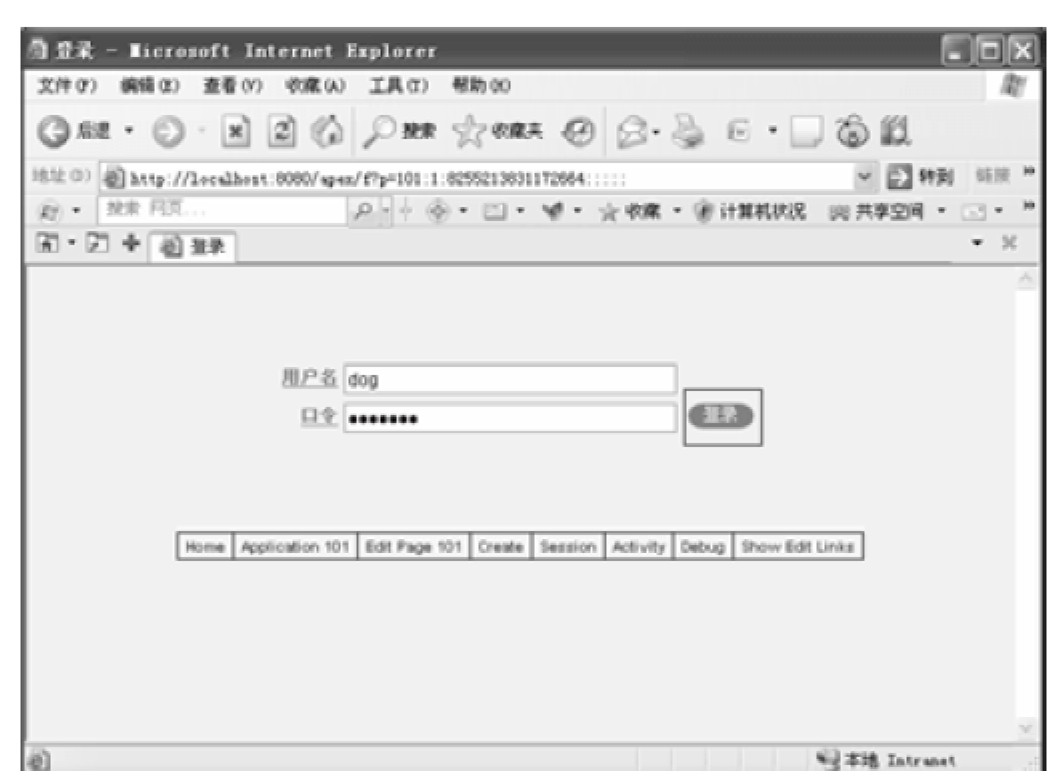


图 22.9

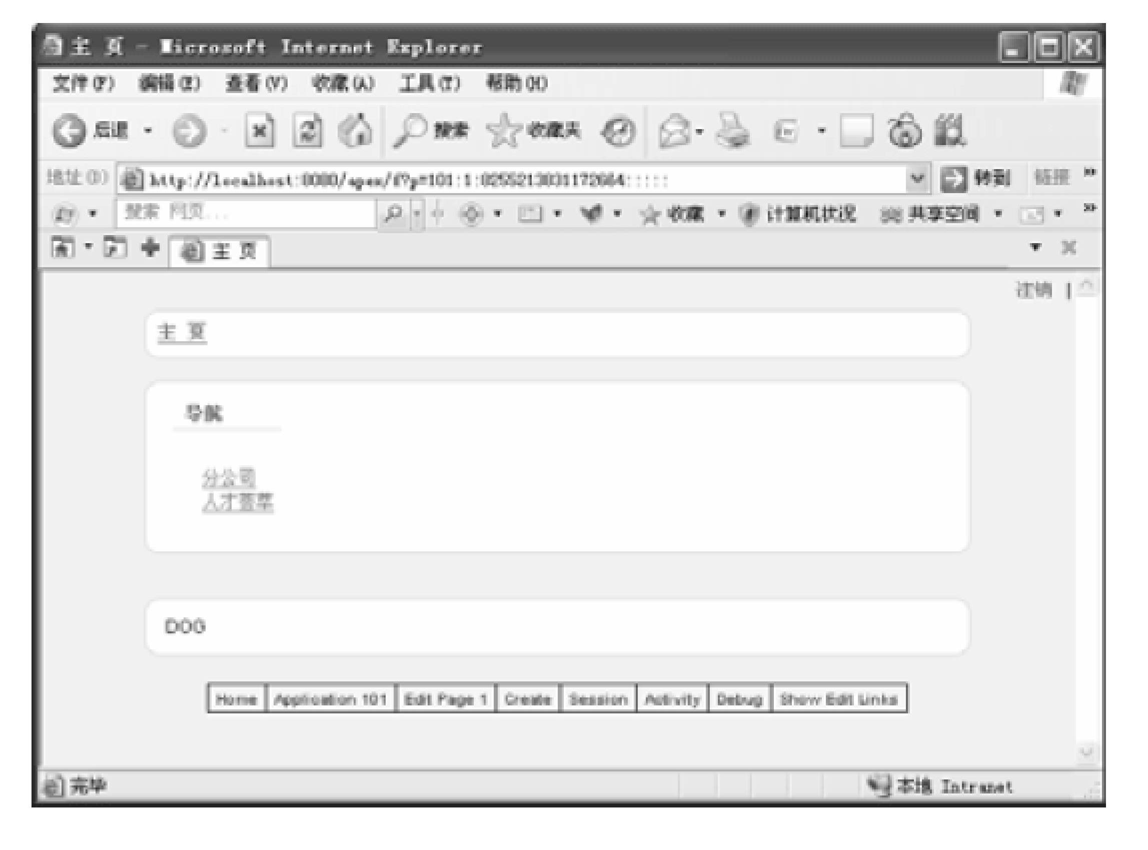
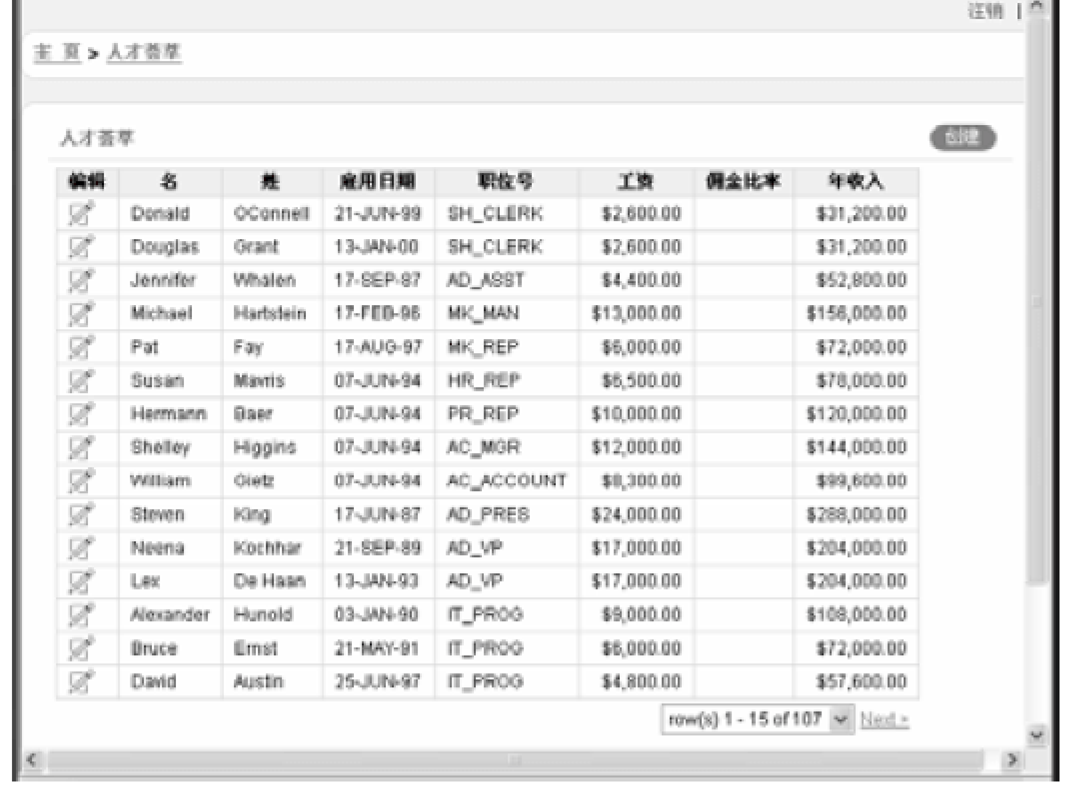


图 22.10



编辑	名	姓	雇用日期	职位号	工资	佣金比率	年收入
	Donald	OConnell	21-JUN-99	SH_CLERK	$2,600.00		$31,200.00
	Douglas	Grant	13-JAN-00	SH_CLERK	$2,600.00		$31,200.00
	Jennifer	Whalen	17-SEP-87	AD_ASST	$4,400.00		$52,800.00
	Michael	Hartstein	17-FEB-96	MK_MAN	$13,000.00		$156,000.00
	Pat	Fay	17-AUG-97	MK_REP	$6,000.00		$72,000.00
	Susan	Mavris	07-JUN-94	HR_REP	$6,500.00		$78,000.00
	Hermann	Baer	07-JUN-94	PR_REP	$10,000.00		$120,000.00
	Shelley	Higgins	07-JUN-94	AC_MGR	$12,000.00		$144,000.00
	William	Gietz	07-JUN-94	AC_ACCOUNT	$8,300.00		$99,600.00
	Steven	King	17-JUN-87	AD_PRES	$24,000.00		$288,000.00
	Neena	Kochhar	21-SEP-89	AD_VP	$17,000.00		$204,000.00
	Lex	De Haan	13-JAN-93	AD_VP	$17,000.00		$204,000.00
	Alexander	Hunold	03-JAN-90	IT_PROG	$9,000.00		$108,000.00
	Bruce	Ernst	21-MAY-91	IT_PROG	$6,000.00		$72,000.00
	David	Austin	25-JUN-97	IT_PROG	$4,800.00		$57,600.00

图 22.11

接下来您将在“人才荟萃”（员工）报表和“分公司”报表之间建立链接，以方便用户的操作。为此，首先要在“人才荟萃”页面上创建一个区域和项，以使用户在该页面上能够选择一个分公司（部门）；然后修改“人才荟萃”报表使其只显示属于所选择分公司（部门）的人才（员工）。其主要操作包括创建区域、创建项、将项与报表链接和创建分支。

22.2 创 建 区 域

在“人才荟萃”页面上创建一个区域以存放分公司（部门）选择列表，具体操作步骤如下：

（1）在“应用程序 101”页面中单击“3-人才荟萃”图标，如图 22.12 所示。之后将出现页 3 的“页定义”界面。

（2）在“区域”栏中单击“创建”图标，如图 22.13 所示。之后将进入“创建区域”界面。

（3）在“标识要添加到此页的区域的类型”栏中接受默认选中的 HTML 单选按钮，然后单击“下一步”按钮，如图 22.14 所示。

（4）在“选择要创建的 HTML 区域容器类型”栏中接受默认选中的 HTML 单选按钮，

然后单击“下一步”按钮，如图 22.15 所示。

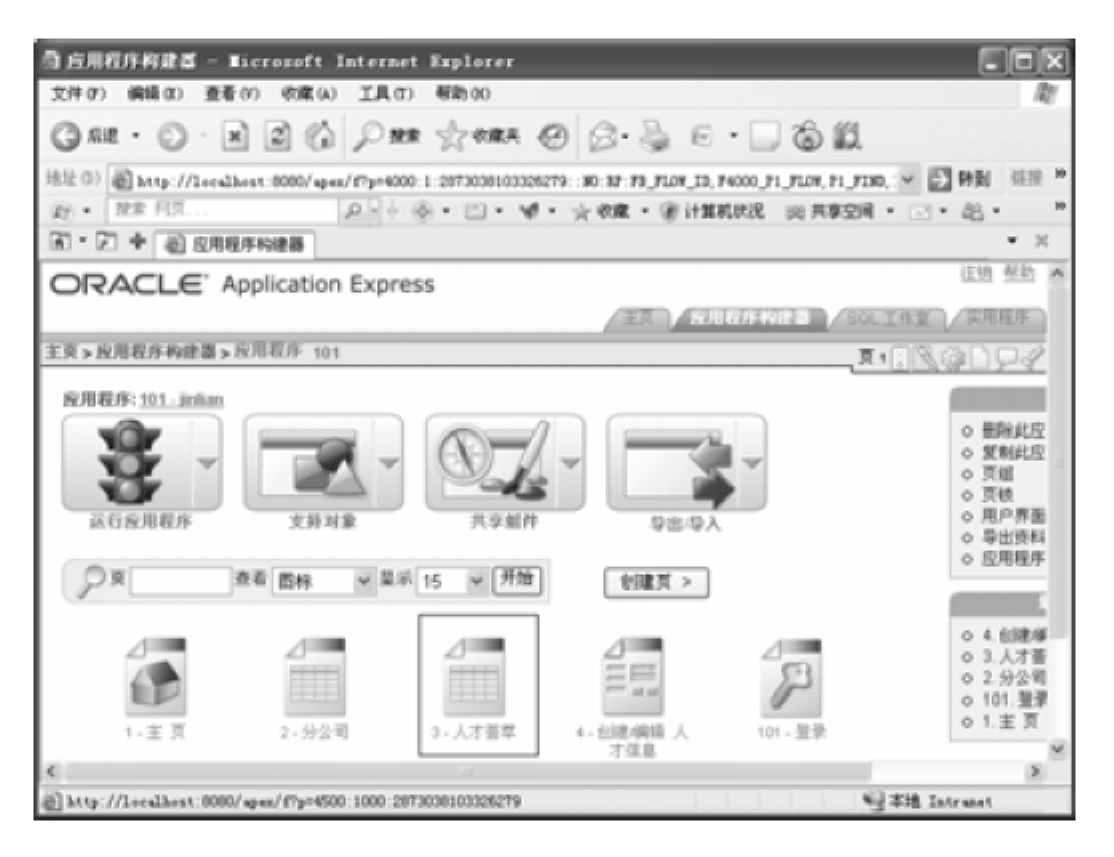


图　22.12



图　22.13



图　22.14

图　22.15

（5）对于显示属性，在“标题”文本框中输入“分公司”，在“区域模板”下拉列表框中选择“无模板”（选项添加了一个没有区域标题的区域）选项，在“序列”文本框中输入“5”（这样该区域将显示在“人才荟萃”超链接之上）。其他部分接受默认值并单击“下一步”按钮，如图 22.16 所示。

（6）单击“创建区域”按钮，如图 22.17 所示。



图　22.16



图　22.17

之后将出现如图 22.18 所示的页面，在“区域”栏中已经多了一个叫分公司的新 HTML 区域，而且序列号为 5，这表明您已经成功地创建了一个区域。



图 22.18

22.3 创 建 项

接下来您将在分公司区域中创建一个项，该项是一个使用分公司值列表的选择列表，具体操作步骤如下：

（1）在“项”栏中单击“创建”图标，如图 22.19 所示。

（2）在“选择项类型”栏中选中“选择列表”单选按钮，之后单击“下一步”按钮，如图 22.20 所示。



图 22.19



图 22.20

（3）在“选择列表控件类型”栏中选中“带有提交功能的选择列表”单选按钮，之后单击“下一步”按钮，如图 22.21 所示。“带有提交功能的选择列表”选项的功能是，无论什么时候，只要选择列表发生变化就要自动地刷新该网页。

（4）对于位置和名称，在“项名”文本框输入“P3_DEPARTMENT_ID”，“序列”接受默认值 10，在“区域”下拉列表框中选择“分公司（1）5”选项，然后单击“下一步”按钮，如图 22.22 所示。



图　22.21



图　22.22

（5）对于值列表，在“命名的 LOV”下拉列表框中选择 DEPARTMENTS 选项（因为您创建 DEPARTMENTS 作为共享组件，所以它出现在选择列表中，这样就能够使您在其他页中使用它），在“显示空值选项”下拉列表框中接受默认值“是”，在“空文本”文本框中输入“未分配部门”，在“空值”文本框中输入“-1”（输入一个空值将使书写查询语句和输入默认值变得容易。如果定义了空值，不管什么时候只要用户选择了“未分配部门”，Express 就将该项的会话状态设置为-1，而且可以在查询语句中使用这一值），然后单击“下一步”按钮，如图 22.23 所示。

（6）对于项属性，在“标签”文本框中输入“分公司”，单击“下一步”按钮，如图 22.24 所示。



图　22.23



图　22.24

（7）对于源，在“项源值”列表框中输入“-1”，其他部分保持原来的默认值，单击“创建项”按钮，如图 22.25 所示。

之后将出现如图 22.26 所示的页面，在“项”栏中已经多了一个新项，而且序列号为 10，这表明您已经成功地创建了一个项。



图　22.25



图　22.26

22.4　将项与报表链接

虽然您已经创建了所需的项，但是该项并未与报表链接。要将其链接到报表上，您必须编辑区域源并加入相关的 WHERE 子句，具体操作步骤如下：

（1）在“区域”栏中单击“人才荟萃”超链接，如图 22.27 所示。之后会出现“编辑区域”页面。

（2）使用最右面的滚动条向下滚动到源部分，如图 22.28 所示。



图　22.27

图　22.28

（3）在区域源中现有的 SQL 代码的最后加入如下的 WHERE 子句，如图 22.29 所示。

```
WHERE (DEPARTMENT_ID = :P3_DEPARTMENT_ID or
(DEPARTMENT_ID is null and nvl(:P3_DEPARTMENT_ID,'1') = '-1'))
```

这个 WHERE 子句使查询语句的结果只显示那些属于所选择的部门（分公司）的员工（人才）的信息。

☞指点迷津：

为了减轻初学者的学习难度，对于比较复杂的命令都在随书的光盘上给出了相关的文件，读者只要在光盘中进入相关的目录（如本章的文件存放在光盘\SQL_Express\AVI\ch22 目录）就可以找到相关的文件，打开文件，将文件中的命令复制到相关的位置就行了。

（4）单击“应用更改”按钮提交所做的编辑修改，如图 22.30 所示。



图　22.29

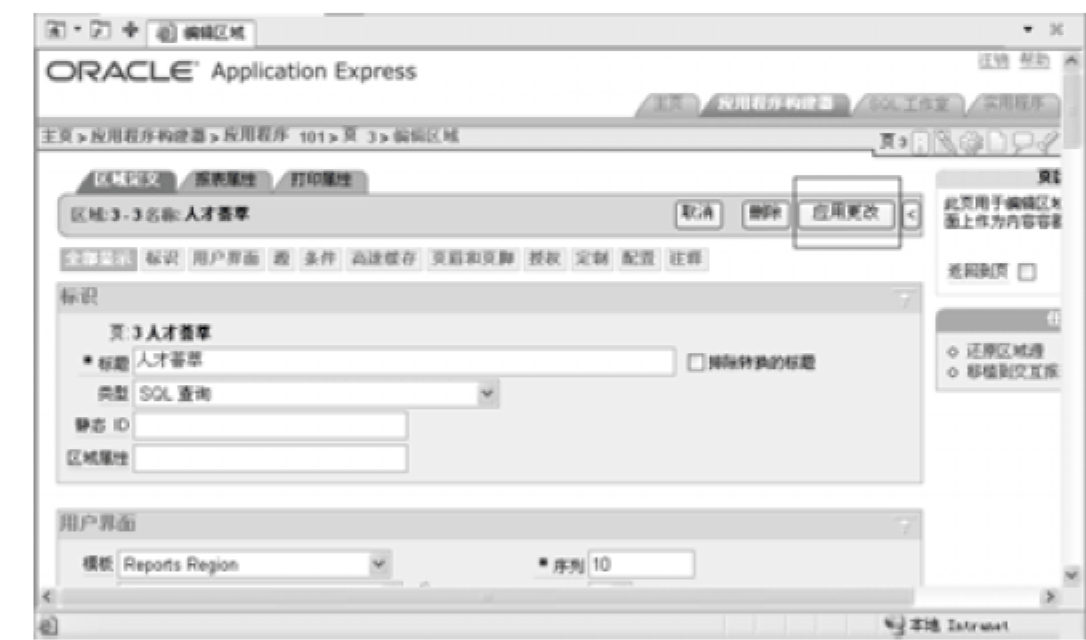


图　22.30

之后就又回到了页 3 的“页定义”页面，如图 22.31 所示。

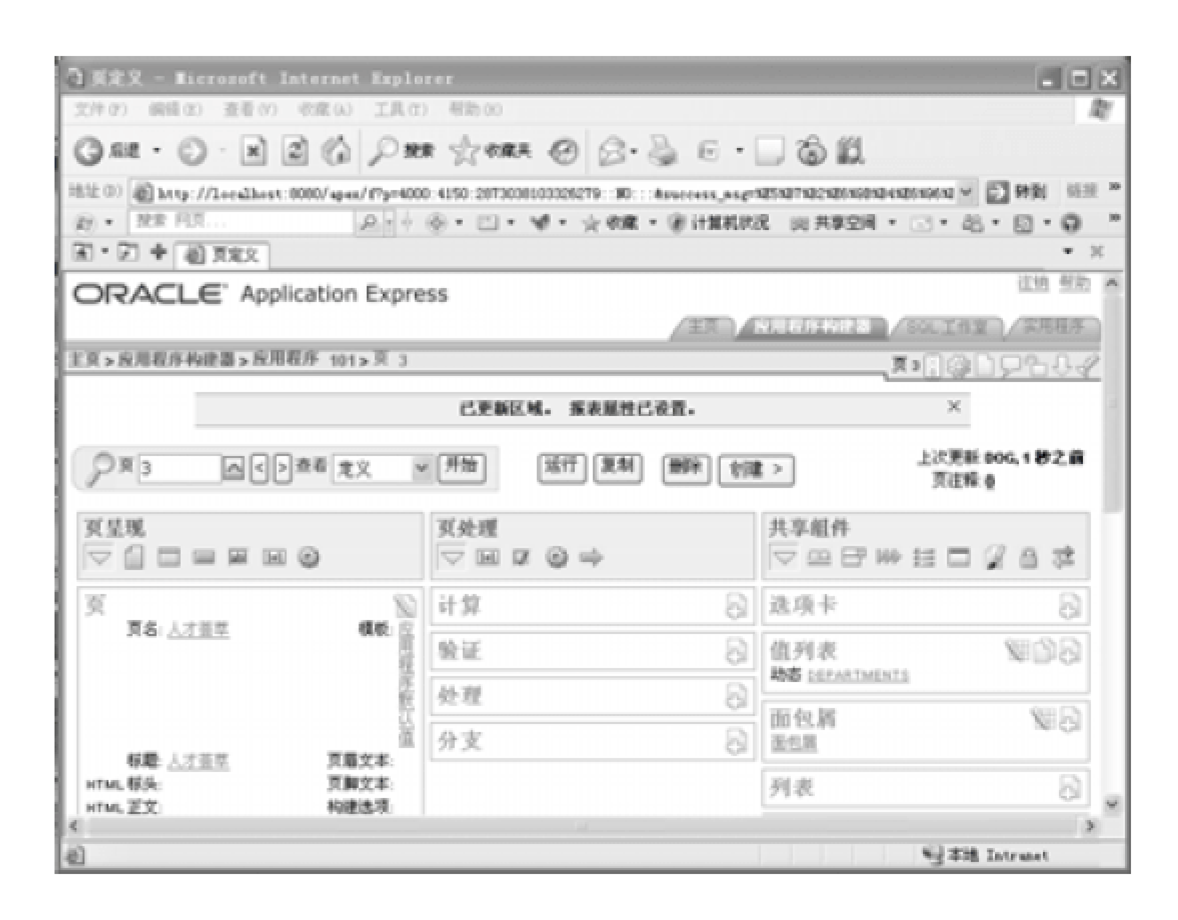


图　22.31

22.5　创 建 分 支

当一个网页被提交时，将由为该页所定义的分支来决定接下来显示哪一页。因为您想让一个用户提交该页时重新显示这一页，所以创建一个指向同一页的分支，具体操作步骤如下：

（1）在“页处理”区域的“分支”栏中单击“创建”图标，如图 22.32 所示。之后将进入“创建分支”的页面。

（2）对于点和类型，接受所有的默认值并单击“下一步”按钮，如图 22.33 所示。

（3）对于目标，在“页”文本框中输入“3”，并选中“重置此页的页码”复选框（当做了这个选择之后，应用程序显示的结果为满足用户查询的第 1 组数据。否则，如果用户在数据的第 3 页并选择另一部门，用户看到的是修改后的查询结果的第 3 页），单击“下一步”按钮，如图 22.34 所示。

（4）单击“创建分支”按钮，如图 22.35 所示。



图 22.32

图 22.33



图 22.34



图 22.35

之后就又回到了页 3 的“页定义”页面，如图 22.36 所示。可以看到在分支部分新增加了一个刚创建的分支。

（5）单击“运行页”图标运行该页，如图 22.37 所示。



图 22.36



图 22.37

（6）打开登录界面，在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”，之后单击“登录”按钮，如图 22.38 所示。

之后将出现类似如图 22.39 所示的页面，在“人才荟萃”列表中会只显示那些属于所选择的分公司的员工（人才），在这里选择的是“未分配部门”选项。

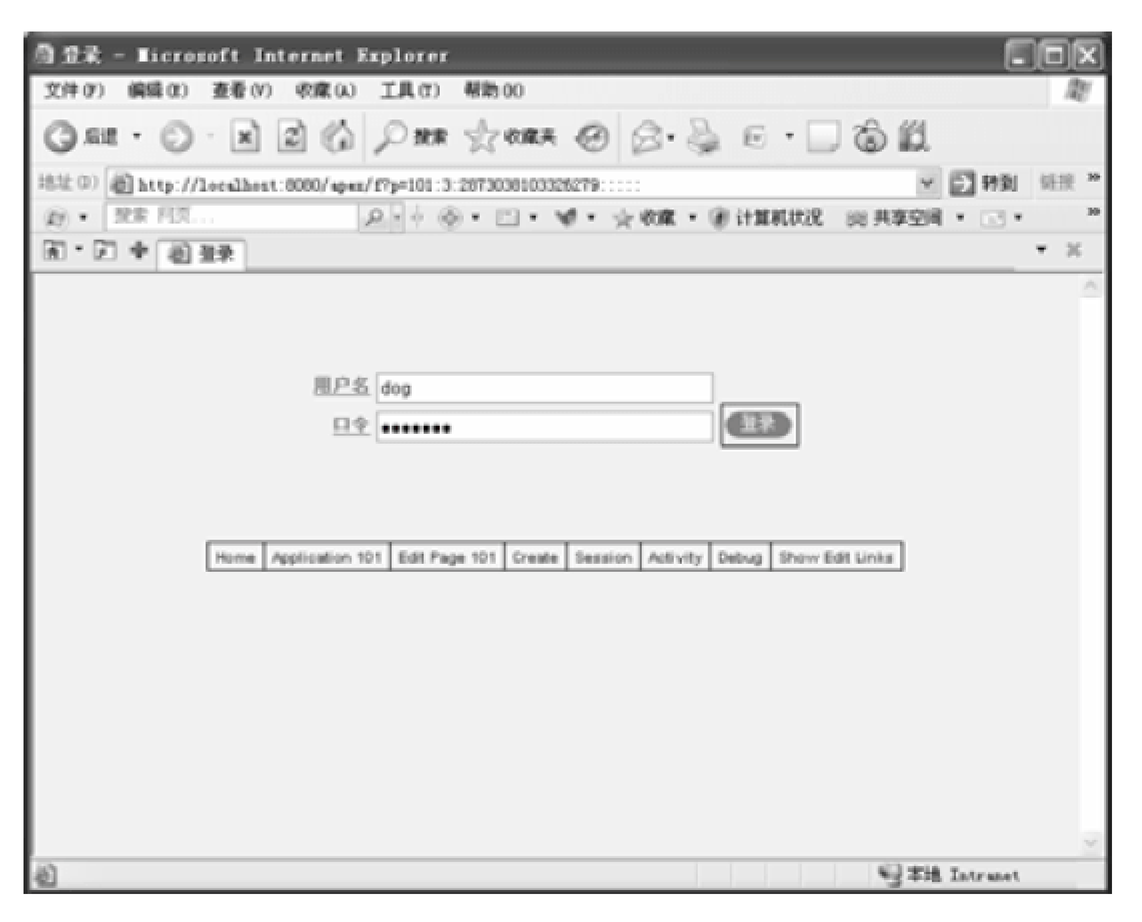


图　22.38



图　22.39

（7）从“分公司”下拉列表框中选择 IT 选项，将会获得所有属于 IT 分公司（部门）的人才（员工）的信息，如图 22.40 所示。



图　22.40

22.6　将一列的值链接到另一个网页

在这一节中，您将在分公司报表上加入一个超链接，该超链接将用户导航到“人才荟萃”报表。该超链接还要将焦点设置在所选择的分公司上。为了完成这些操作，您需要编辑“分公司”报表上的一些报表属性。以下就是将列值修改成一个超链接的具体操作步骤：

（1）在面包屑处单击“主页”超链接以退回到主页，如图 22.41 所示。

（2）在导航部分单击“分公司”超链接以进入分公司页，如图 22.42 所示。

（3）在开发工具栏中单击 Edit Page 2 超链接，如图 22.43 所示。之后将进入页 2 的“页定义”页面。

（4）在“区域”栏中单击“报表”超链接，如图 22.44 所示。之后将进入页 2 的“报表属性”页面。



图 22.41

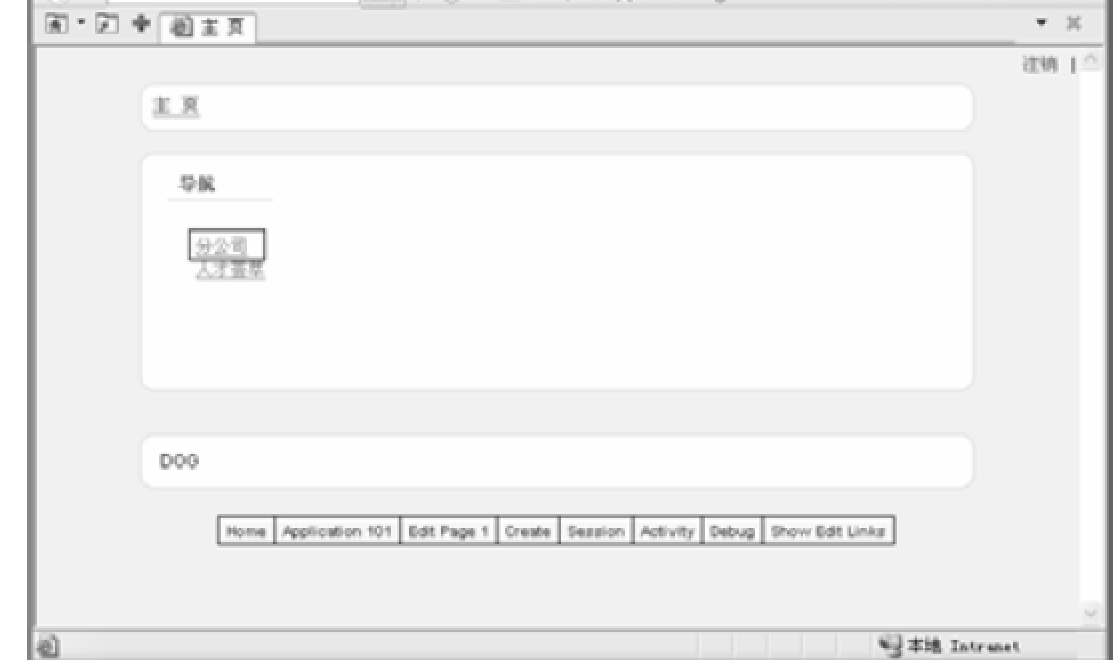
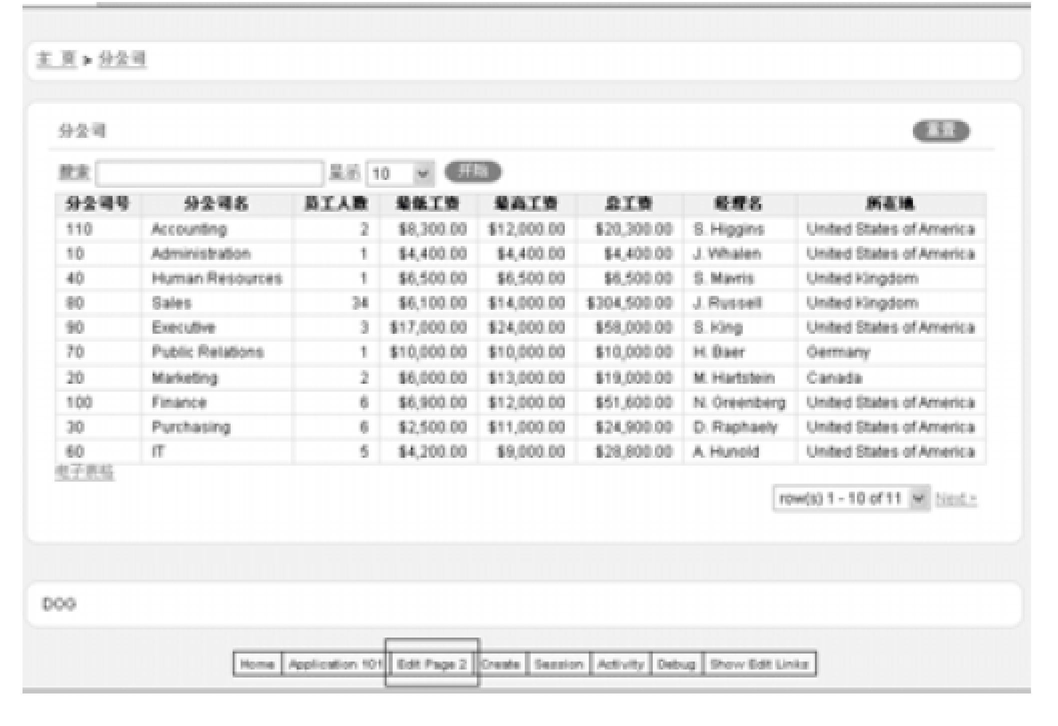


图 22.42



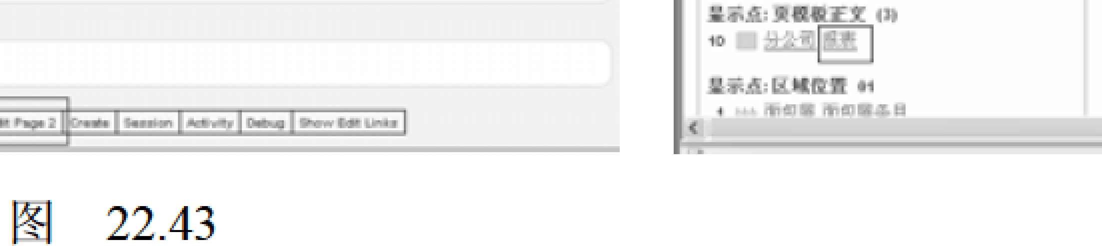


图 22.43

图 22.44

（5）找到列属性部分，并将员工人数列改为右对齐，如图 22.45 所示。

（6）单击“员工人数”左面的编辑图标，如图 22.46 所示。之后将进入“列属性”页面。



图 22.45



图 22.46

（7）向下滚动最右面的滚动条直到“列链接”区域，如图 22.47 所示。

（8）单击“链接文本”文本框右面的图标，如图 22.48 所示。之后将出现“选取列”页面。

（9）选择“#员工人数#”选项作为链接文本，如图 22.49 所示。

（10）在“链接属性”文本框中输入“alt="浏览人才信息" title="浏览人才信息"”，在“页”文本框中输入“3”，选中“重置页码”复选框，将项 1 的名称设置为 P3_DEPARTMENT_ID，将项 1 的值设置为“#分公司号#”，如图 22.50 所示。



图 22.47

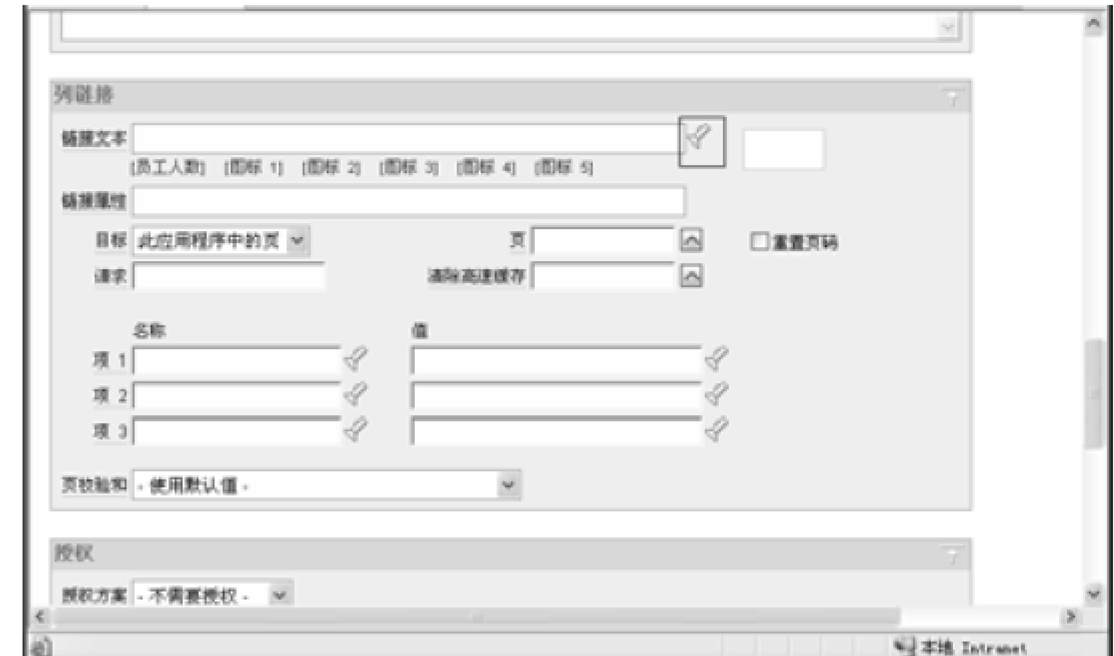


图 22.48

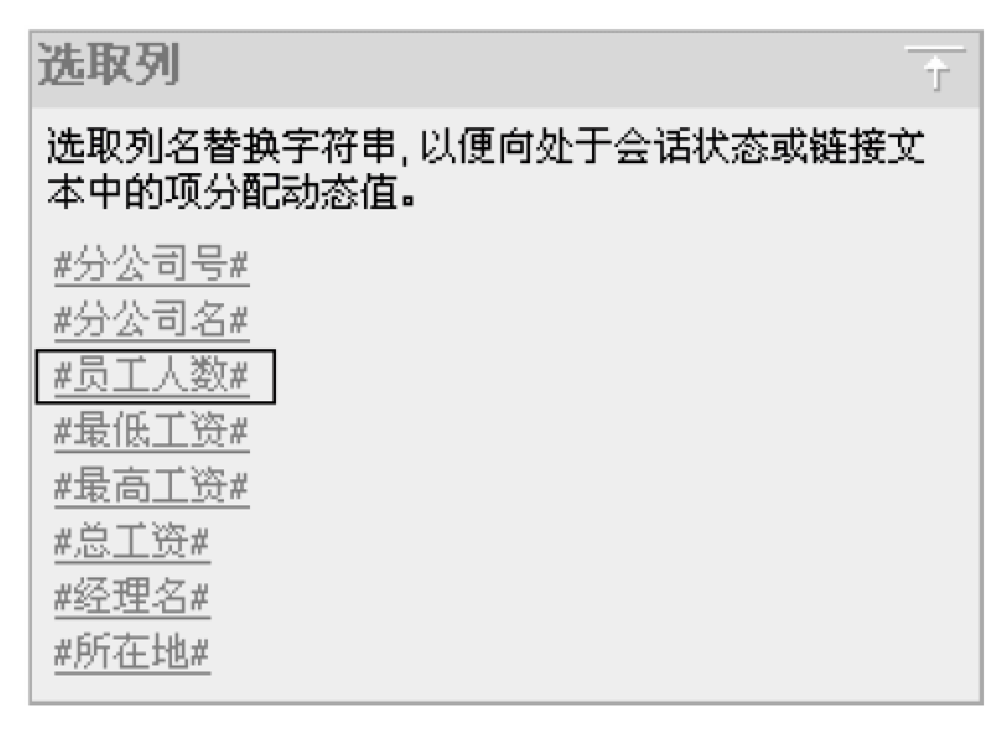


图 22.49



图 22.50

“列链接”区域显示了可以选择的所有选项，当终端用户单击该链接时这些选项能够完成如下的操作：

- 将终端用户导航到第 3 页“人才荟萃”报表。
- 在“人才荟萃”报表上聚焦在所选择的分公司上。
- 当用户选择另一分公司（部门）时，返回给用户的是该分公司的第 1 组人才（员工）记录。

（11）单击“应用更改”按钮，如图 22.51 所示。之后将返回“报表属性”页面。

您可以看到在图 22.52 中的“列属性”区域，“员工人数”行在“链接”列中有一个对号，这表明您已经对其进行了修改。

（12）单击“运行页”图标以运行该页，如图 22.52 所示。



图 22.51



图 22.52

在“分公司”报表中，每一行数据的“员工人数”值都有一个下划线，如图 22.53 所示。现在终端用户可以通过这些超链接来向下了解更详细的信息。

（13）在分公司号为 100 的 Finance 分公司（也可以选择其他的分公司）记录上单击“员工人数”超链接以测试所创建的链接，如图 22.54 所示。



图 22.53

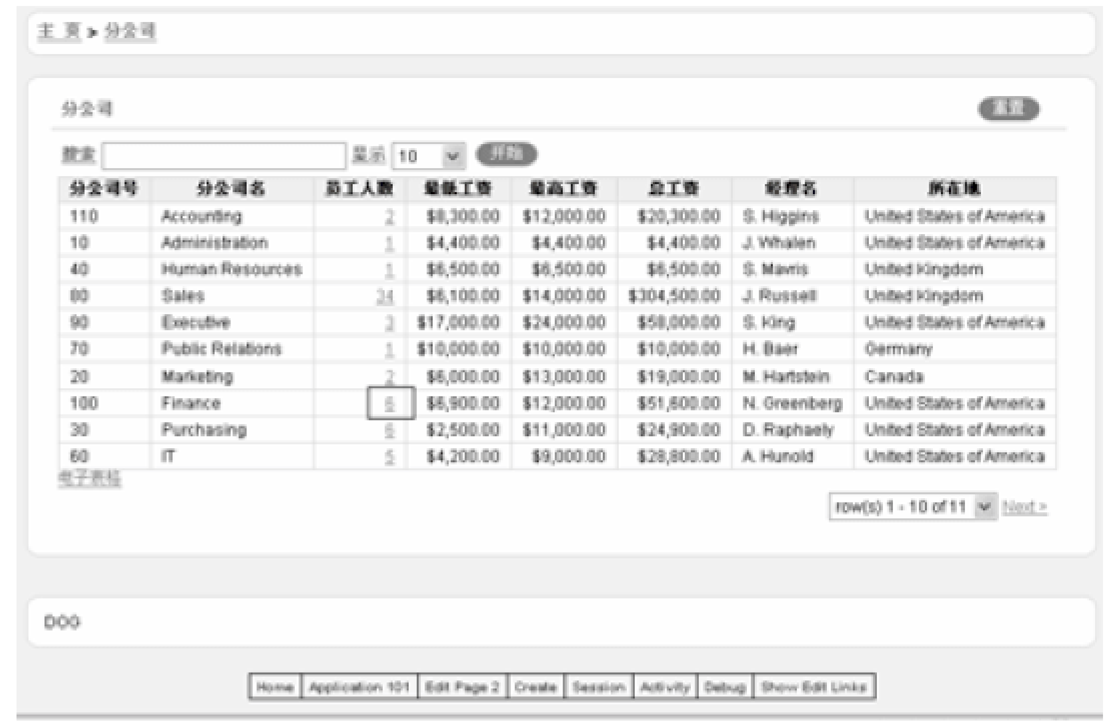


图 22.54

之后就会获得只属于所选择的分公司 Finance 的“人才荟萃”（员工）报表的信息，如图 22.55 所示。

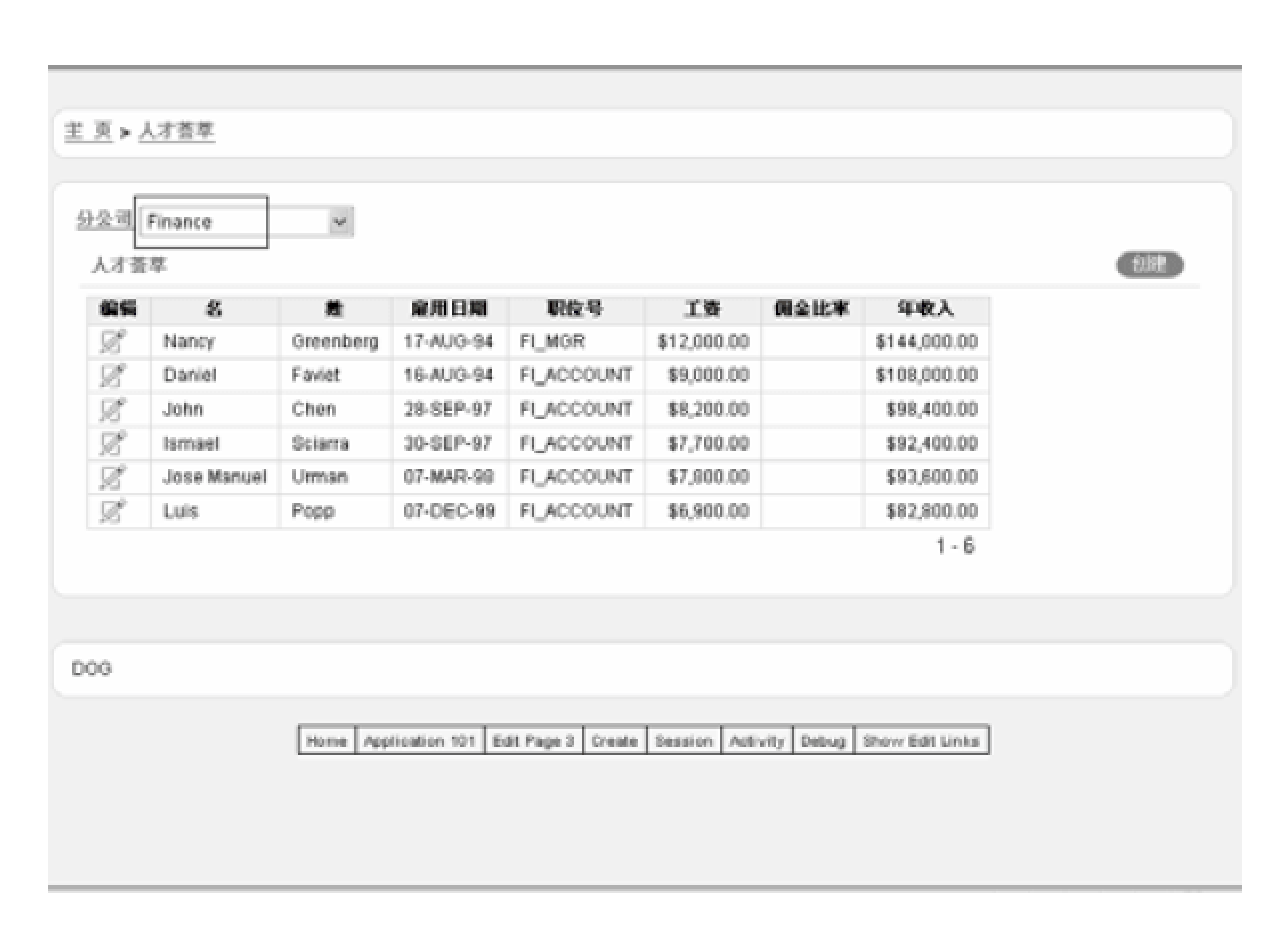


图 22.55

接下来您要增加一个作为显示分公司（部门）细节的报表，再创建一个条件以便增加的报表只显示您选择的分公司（部门）的信息即可。如果您选择了“未分配部门”，将不会显示分公司（部门）细节报表。要实现以上的功能，首先要创建报表和条件，然后再关闭页码。

22.7 创建报表和条件

首先，创建报表并设置报表显示的条件，具体操作步骤如下：

（1）单击开发工具栏中的 Edit Page 3 超链接以导航到页 3 的“页定义”页面，如

图 22.56 所示。

（2）在“区域”栏中单击“创建”图标，如图 22.57 所示。之后将进入“创建区域”页面。

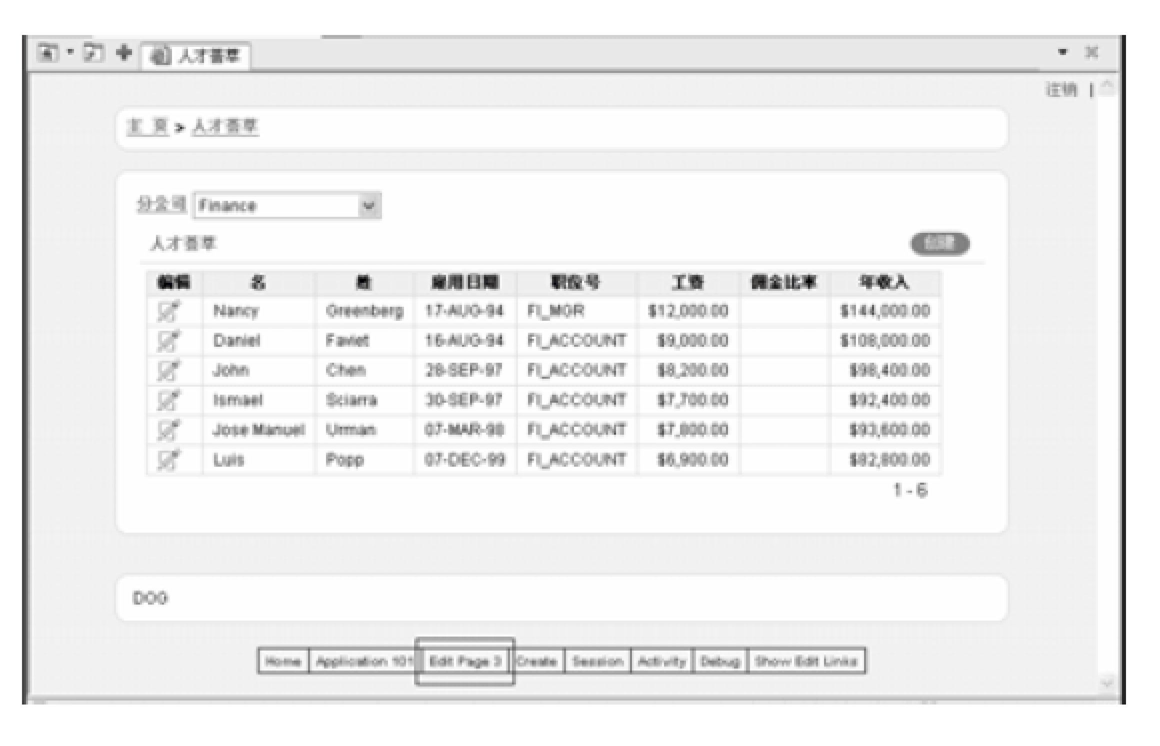


图　22.56



图　22.57

（3）在“标识要添加到此页的区域的类型”栏中选中“报表”单选按钮，然后单击“下一步”按钮，如图 22.58 所示。

（4）在“报表实施”栏中接受默认选中的“SQL 报表”单选按钮，并单击“下一步”按钮，如图 22.59 所示。



图　22.58



图　22.59

（5）对于显示属性，在“标题”文本框中输入“分公司细节”，在“区域模板”下拉列表框中选择“无模板”选项，在“序列”文本框中输入“7”，其他接受默认设置并单击“下一步”按钮，如图 22.60 所示。

（6）对于源，在“输入 SQL 查询或返回 SQL 查询的 PL/SQL 函数”列表框输入如下的 SQL 语句（如图 22.61 所示）。

```
SELECT count(e2.employee_id) "员工人数:",
substr(e.first_name,1,1)||'. '|| e.last_name "经理姓名:",
c.country_name "地点:"
FROM departments d, employees e,
locations l, countries c,
employees e2
```

```
WHERE d.manager_id = e.employee_id
AND d.location_id = l.location_id
AND d.department_id = e2.department_id
AND l.country_id = c.country_id
AND nvl(d.department_id,'-1') = nvl(:P3_DEPARTMENT_ID,'-1')
GROUP BY substr(e.first_name,1,1)||'. '||e.last_name, c.country_name
```

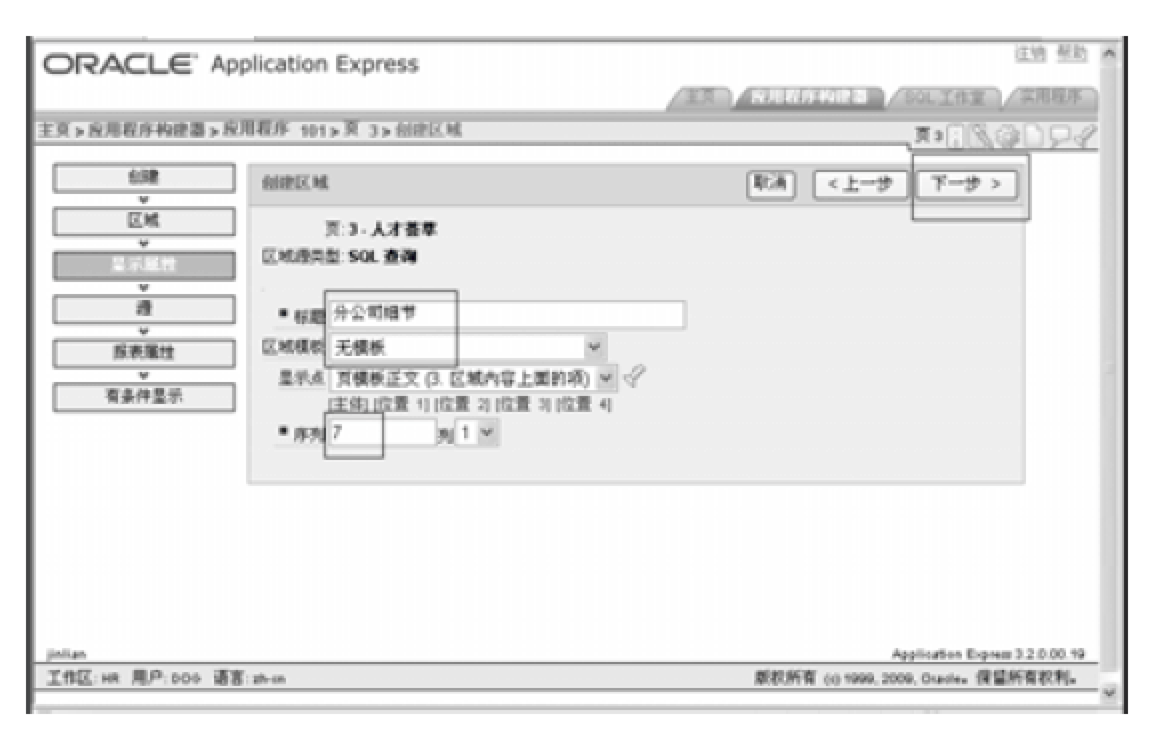


图 22.60



图 22.61

（7）单击“下一步”按钮，如图 22.62 所示。

通过以上的操作，您已经创建了一个标题为“分公司细节”的报表。接下来您将创建一个条件，该条件在用户选择“未分配部门”选项时将不显示分公司（部门）细节信息。

（8）对于报表属性，在“报表模板”下拉列表框中选择“默认：垂直报表，外观 1（包含空列）”选项（这一选择显示垂直列表），其他保持默认设置并单击“下一步”按钮，如图 22.63 所示。



图 22.62

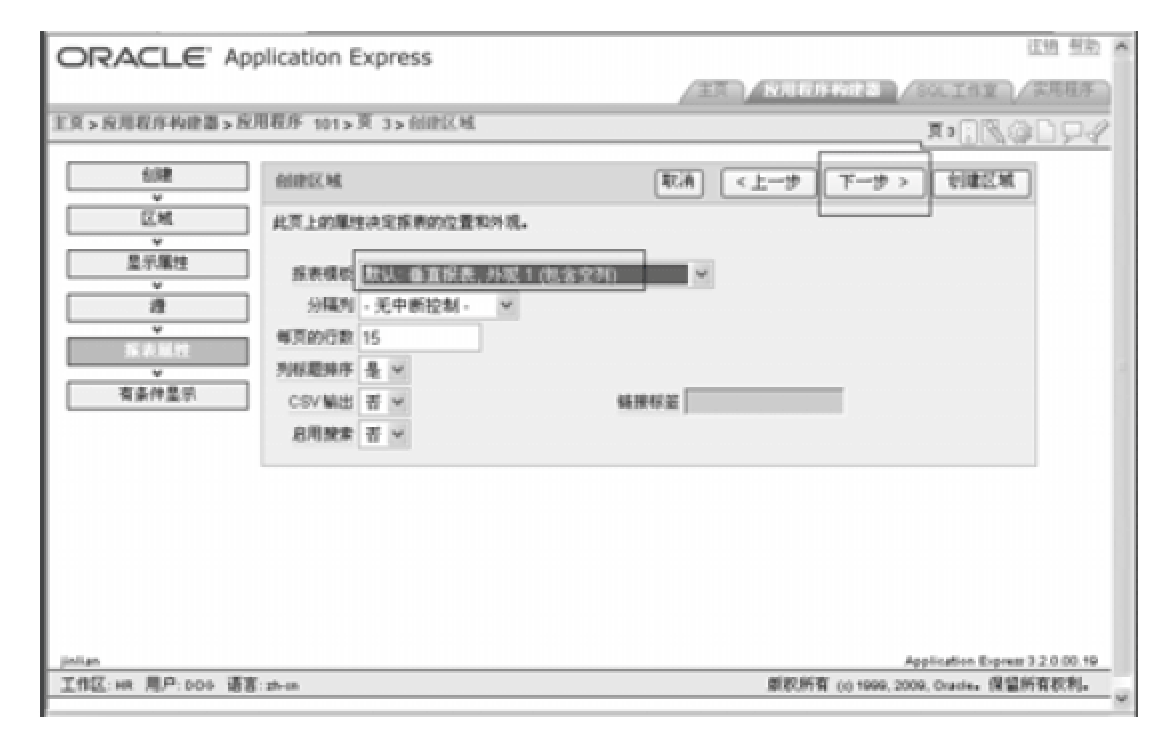


图 22.63

（9）对于有条件显示，在“条件类型”下拉列表框中选择“表达式 1 中的项值！=表达式 2”选项，在“表达式 1”列表框中输入“P3_DEPARTMENT_ID”、在“表达式 2”列表框中输入“-1”，单击“创建区域”按钮，如图 22.64 所示。

之后将返回页 3 的“页定义”页面，在“区域”栏中出现了一个新的区域，这个区域就是您刚创建的，如图 22.65 所示。



图　22.64



图　22.65

（10）单击“运行页”图标，如图 22.66 所示。

之后将出现如图 22.67 所示的页面，分公司的细节信息将出现在网页的左上方。您可能已经注意到了，在分公司细节下出现了 1-1 的页码。由于一个公司的细节信息只能有一个记录，所以页码的显示是完全没有必要的。



图　22.66



图　22.67

22.8　关 闭 页 码

在这一节中，您将关闭页码，以下是具体的操作步骤：

（1）在页 3 的“页定义”页面的“区域”栏中单击“报表”（“公司细节”右侧）超链接，如图 22.68 所示。之后将进入“报表属性”页面。

（2）使用最右面的滚动条向下滚动到“布局和页码”区域，如图 22.69 所示。

（3）在“页码方案”下拉列表框中选择“-未选择页码-”选项，如图 22.70 所示。

（4）单击“应用更改”按钮，如图 22.71 所示。

（5）单击“运行页”图标运行该网页，如图 22.72 所示。之后将出现“人才荟萃”页面。

由于此时显示的分公司是“未分配部门”，所以没有显示分公司的细节信息，如图 22.73

所示。



图 22.68

图 22.69



图 22.70

图 22.71



图 22.72

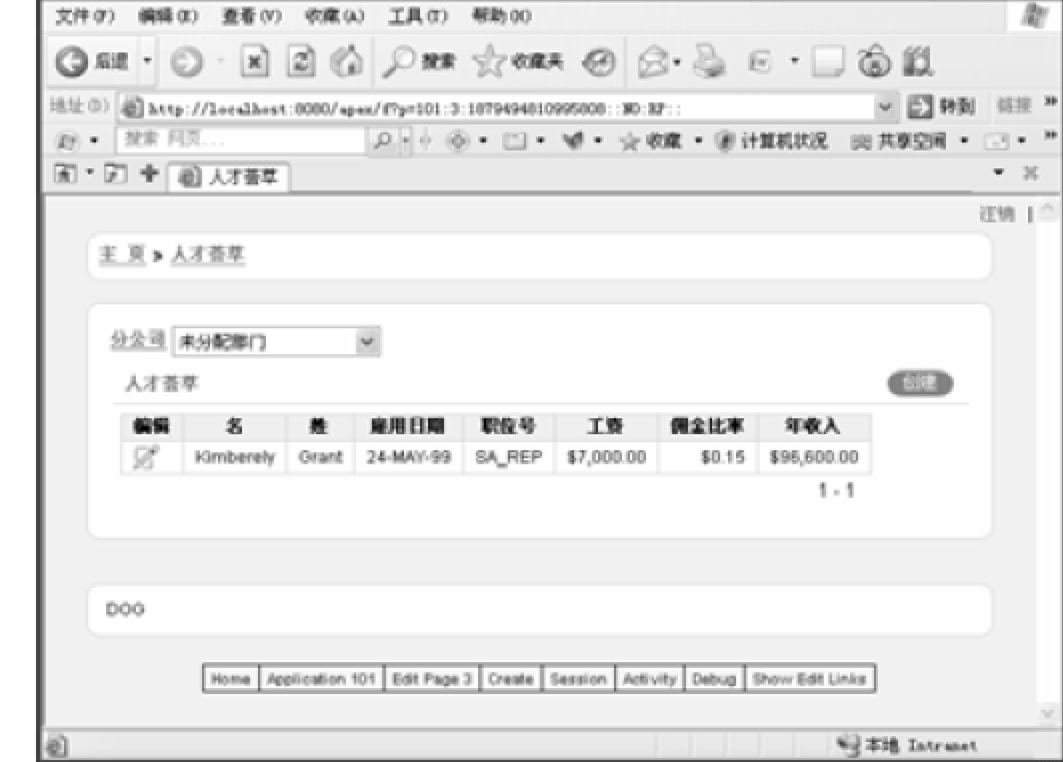


图 22.73

（6）在“分公司”下拉列表框中选择 Accounting 选项，之后将出现该分公司（Accounting）的细节信息，而且也没有页码了，如图 22.74 所示。

接下来根据老板的嘱咐，您还要将公司的广告用语“武大郎驴肉火烧~一个飘香千年的中华品牌、一段流传千古的爱情传奇！！！”加到每一页的最上面。



图　22.74

22.9　添加广告用语

利用 Express 可以在应用程序中添加一个图形化的标记或文本，使它出现在应用程序的每个网页上。页模板决定这一标记或文本显示的位置。在这一节中，您将把公司的广告用语“武大郎驴肉火烧~一个飘香千年的中华品牌、一段流传千古的爱情传奇！！！”加到刚开发的应用程序上，以下就是具体的操作步骤：

（1）在面包屑中单击“主页”超链接以返回主页，如图 22.75 所示。

（2）在最底部的开发工具栏中单击 Application 101 超链接，如图 22.76 所示。

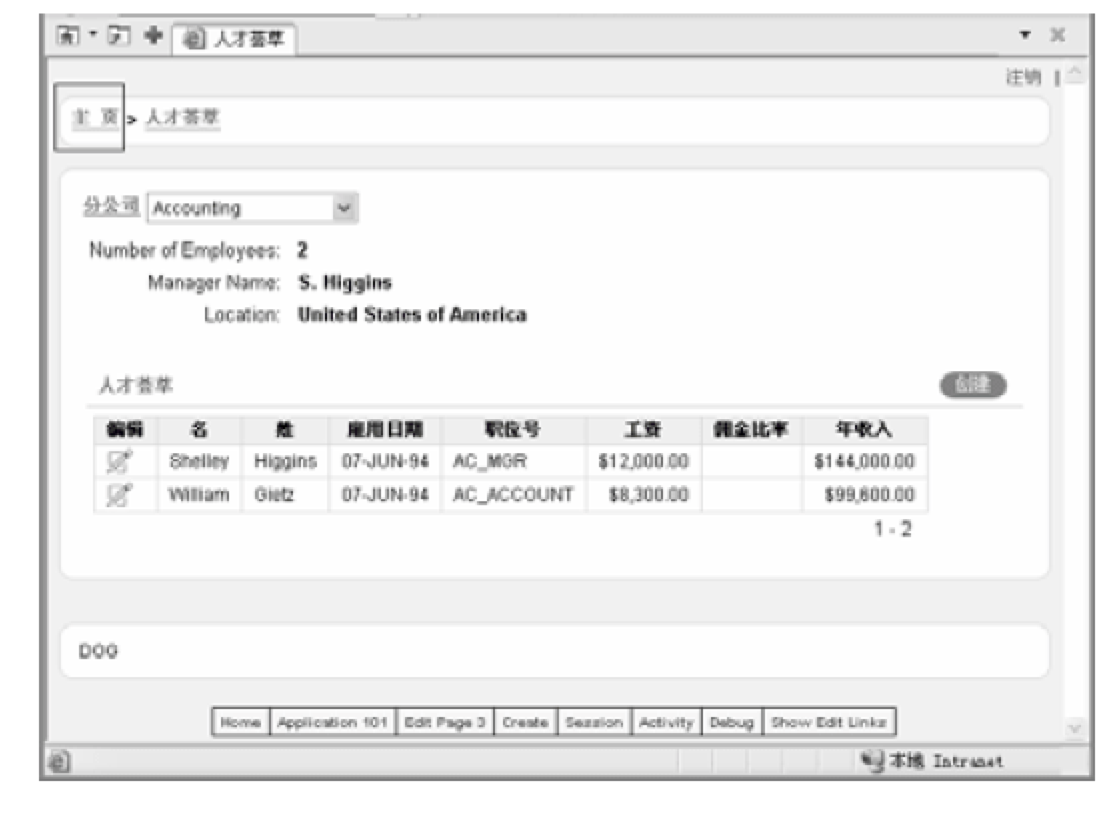


图　22.75

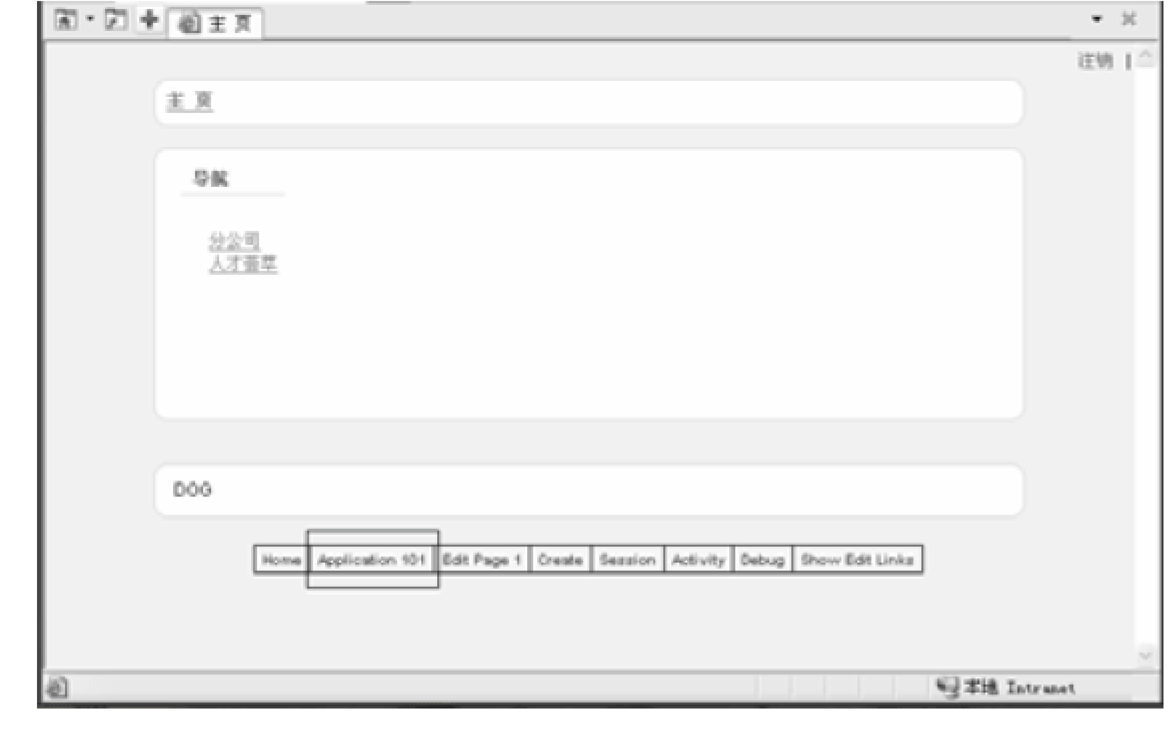


图　22.76

（3）单击“共享组件”图标，如图 22.77 所示。

（4）在“应用程序”栏中单击“定义”超链接，如图 22.78 所示。之后将进入“编辑应用程序定义”页面。

（5）在“徽标类型”栏中选中“文本”单选按钮，在“徽标”文本框中输入“武大郎驴肉火烧~一个飘香千年的中华品牌、一段流传千古的爱情传奇！！！”，在“徽标属性”下拉列表框中选择“黑色文本”选项，如图 22.79 所示。

（6）单击“应用更改”按钮，如图 22.80 所示。之后返回“共享组件”页面。



图 22.77



图 22.78



图 22.79



图 22.80

（7）单击右上角的“运行页”图标运行应用程序，如图 22.81 所示。

之后将出现类似图 22.82 所示的页面。“武大郎驴肉火烧~一个飘香千年的中华品牌、一段流传千古的爱情传奇！！！”这一响亮的口号将出现在应用程序的每一个网页上。



图 22.81

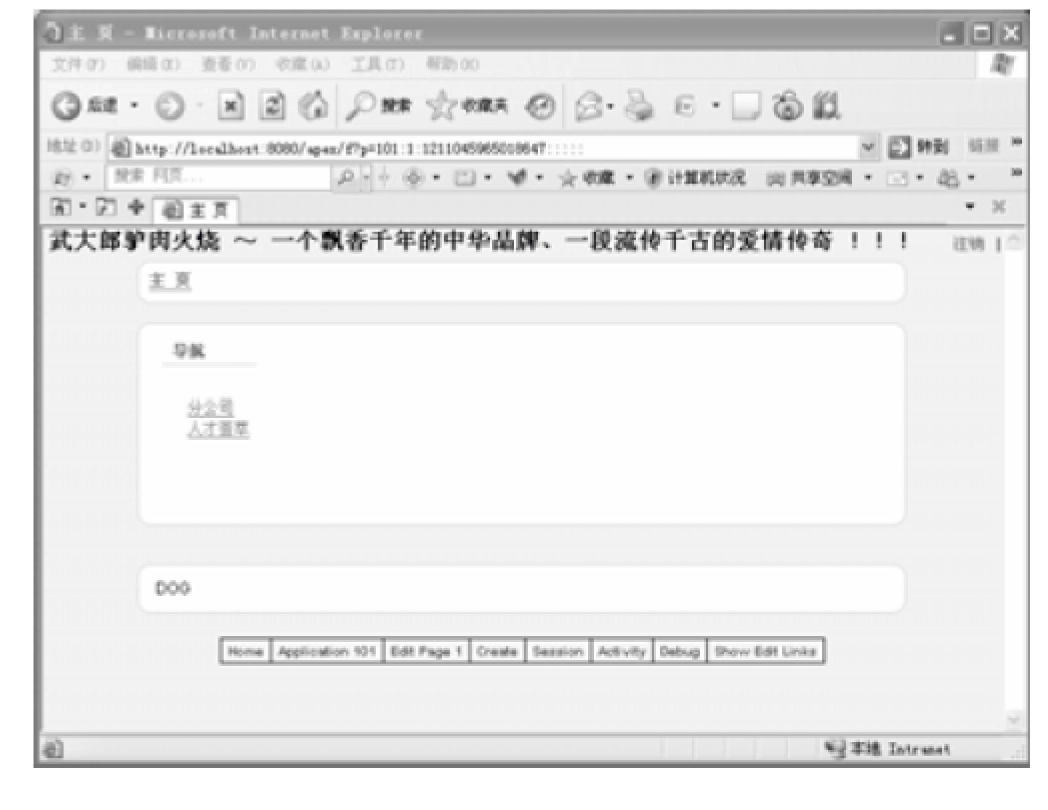


图 22.82

22.10 如何使用“发现”图标

到目前为止，您已经创建了许多应用程序的组件。现在一定很想知道到底自己有多少

“家底”，有没有一种方便、快捷的方法来获取这些信息呢？Oracle这么伟大的设计肯定早已高瞻远瞩地想到了这一点。Oracle Application Express在应用程序构建器中的多数页面上都有一个“发现”图标（手电筒形状），可以通过单击“发现”图标来搜寻项、页、查询语句、表（包括视图）、PL/SQL代码、图像和层叠式样式表（CSS）。

“发现”图标的外形类似一个手电筒，通常在“运行”页面和“编辑”页面的右侧。“发现”图标出现在应用程序构建器中的多数页面上，包括应用程序主页、定义页面、应用程序属性页面和创建与管理共享组件的多数页面上。下面分别进行讲解。

在“应用程序 101”页面单击右上角最右边的“发现”图标，如图 22.83 所示，之后就会进入“项查找器”对话框。

搜索条显示在“项查找器”对话框的上方并且包括了如下的控制。

- 搜索：用于搜索一个项名。在“搜索”文本框中输入大小写无关的关键字，选择页号，之后单击“开始”按钮。为了查看该页的所有项，可以在“搜索”文本框不输入关键字并单击“开始”按钮，如图 22.84 所示。

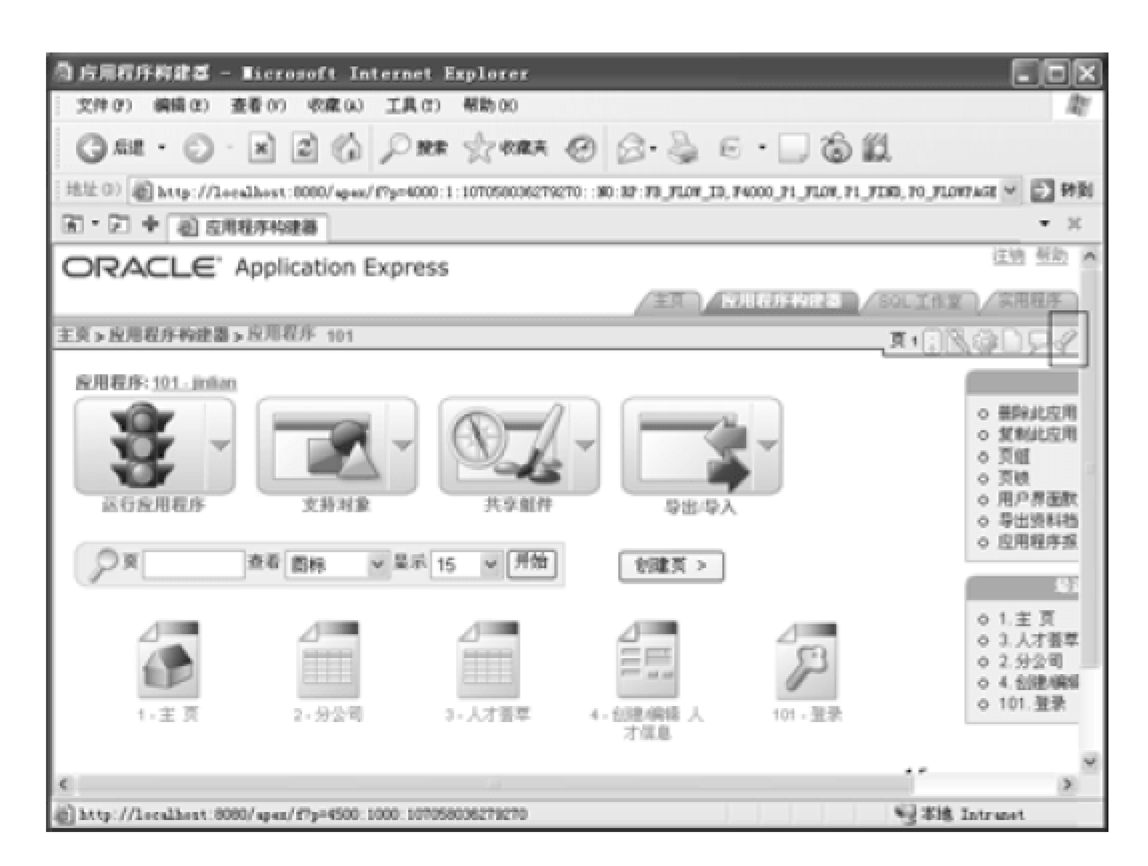


图　22.83



图　22.84

- 页：用于搜索包含项的页。在“页”文本框中输入页号并单击“开始”按钮。要列出该应用程序的所有页，单击“页”文本框右边的“值列表”按钮，将列出所有的页，如图 22.85 所示。
- 显示：用于决定在结果报表中显示的行数。要改变显示的行数，在“显示”下拉列表框中作出选择后单击“开始”按钮即可，如图 22.86 所示。

接下来介绍“项查找器”对话框中各选项卡的功能。

- 选择“页”选项卡，将出现每一页的概要信息，如图 22.87 所示。页是一个应用程序的基本构件，也包括“搜索”栏和相关的控制。
- 选择“查询”选项卡，将出现所定义的每一个 SQL 查询语句和相关页、区域的信息，如图 22.88 所示。
- 选择“表”选项卡，将出现所有表和视图的信息，其中包含每个表和视图中的数据行数及上一次分析的时间，如图 22.89 所示。以 DEMO_开始的表示 Express 自动安装的样本表，可以单击“下一页”按钮查看更多的表和视图。

● 选择 PL/SQL 选项卡，将出现所有的 PL/SQL 程序信息，可以选中“过程”、“函数”或“（软件）包”复选框，如图 22.90 所示。



图 22.85

图 22.86



图 22.87



图 22.88



图 22.89



图 22.90

- 在“名称”列中选择某个程序超链接就会出现该程序的详细的定义信息，如源（程序）类型和接口参数等，如图 22.91 所示。
- 选择“图像”选项卡，将出现所有预安装的图像，这些图像可以在应用程序中直接使用，如图 22.92 所示。有关图像的使用读者可以参阅相关版本的 Application Builder User’s Guide。



图　22.91



图　22.92

（7）选择CSS选项卡，将出现所有的层叠式样式表（CSS），如图 22.93 所示。



图　22.93

22.11　在主页上加入客户信息

当老板看到您所开发的应用程序之后，她觉得还应该将客户的信息放在主页上，同时要为这些信息起一个响亮的名字。一位经理觉得现在媒体上常常将客户比喻成“上帝”，因此他认为应该将这些信息的标题改成“上帝”。

可是老板的一番话却提醒了您。她说：“这洋鬼子脑子就是有问题。他们供的上帝又是

拿钉子钉，又是拿利刃刺，弄得血淋淋的，这供在家里也不吉利呀。再看咱们的佛和菩萨，一个个都是慈眉善目的，看上去就喜性。这鬼子就是虚伪，什么民主，您看天上就一个上帝，那不是独裁是什么。再看咱们的神和菩萨多的是，这才是真的民主。这样也方便，用到谁就拜谁。这鬼子清规戒律也多，每天要做弥撒（礼拜）。现在人都忙得不得了，哪有那么多时间。再看咱们的神，都是什么时候用了什么时候拜。”

透过老板的这些话，您觉得将客户信息的标题改为“活菩萨”可能更合适。当您把这一想法告诉老板时，她说这正是她的意思。以下就是您将这些“活菩萨”的信息放在应用程序主页上的具体操作步骤：

（1）在主页定义页面的“区域”栏中单击“创建”图标，如图 22.94 所示。之后就进入“创建区域”页面。

（2）选中“报表”单选按钮，单击“下一步”按钮，如图 22.95 所示。



图　22.94



图　22.95

（3）选中“SQL 报表”单选按钮，单击“下一步”按钮，如图 22.96 所示。

（4）在“标题”文本框中输入“活菩萨信息”，在“区域模板”下拉列表框中选择 Reports Region 选项，在“序列”文本框中输入 30，之后单击“显示点”下拉列表框最右侧的“发现”图标，如图 22.97 所示。之后会出现“区域显示点”对话框。



图　22.96



图　22.97

（5）在“区域显示点”对话框中单击“区域位置 2”超链接，如图 22.98 所示。之后

会退回到“创建区域”页面。

（6）在“创建区域”页面单击“下一步”按钮，如图 22.99 所示。

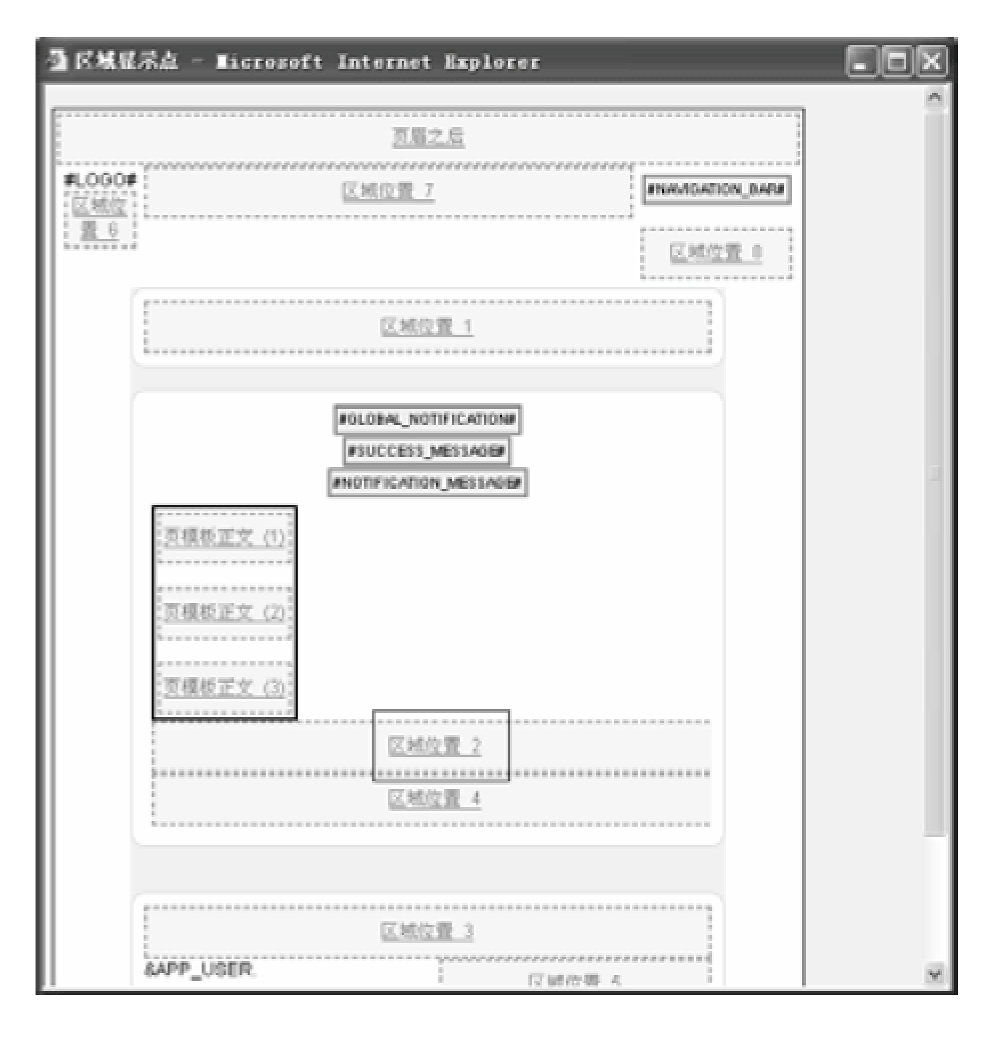


图　22.98



图　22.99

（7）您可以通过单击“查询构建器”按钮让 Express 自动生成所需的 SQL 语句，如图 22.100 所示。之后将进入“查询构建器”页面。

（8）在“查询构建器”对话框的左侧选择 DEMO_CUSTOMERS 表，如图 22.101 所示。



图　22.100

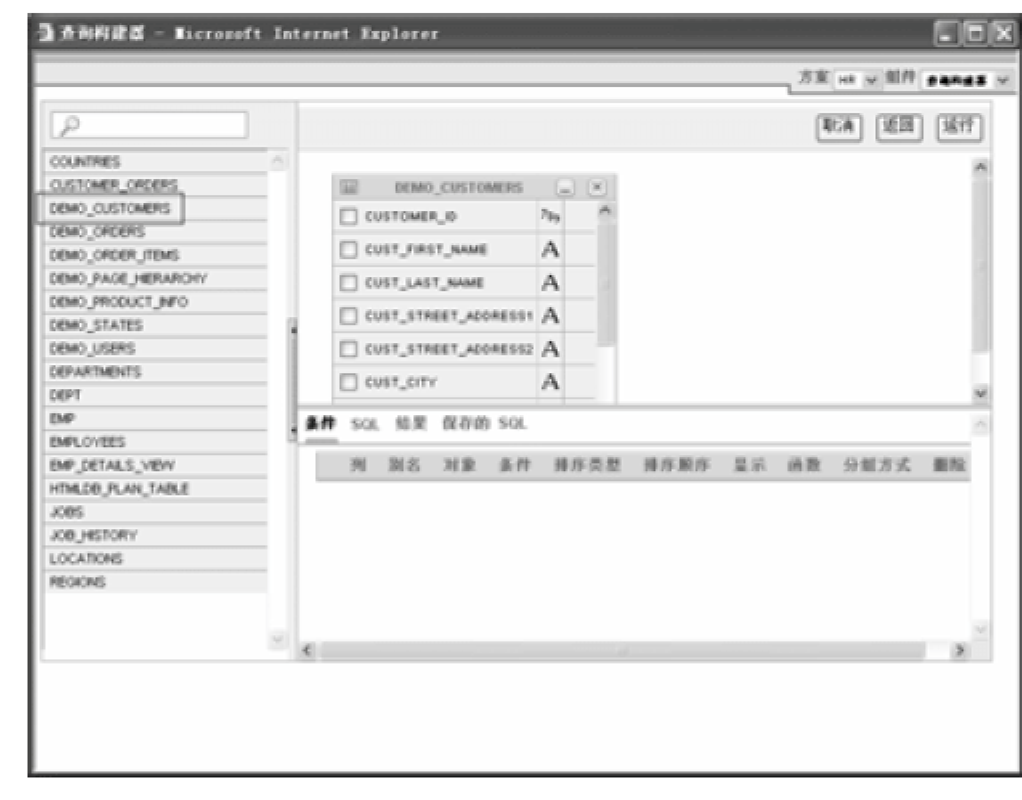


图　22.101

（9）在 DEMO_CUSTOMERS 表中选择 CUSTOMER_ID、CUST_FIRST_NAME、CUST_LAST_NAME、CUST_CITY、PHONE_NUMBER1、CREDIT_LIMIT 和 CUST_EMAIL 列，如图 22.102 所示。

（10）将列 CUSTOMER_ID 的别名改为“客户号”，将列 CUST_FIRST_NAME 的别名改为“客户名”，将列 CUST_LAST_NAME 的别名改为“客户姓”，将列 CUST_CITY 的别名改为“城市”，将列 PHONE_NUMBER1 的别名改为“电话”，将列 CREDIT_LIMIT 的别名改为“信用额”，将列 CUST_EMAIL 的别名改为“电子邮件”，如图 22.103 所示。



图 22.102

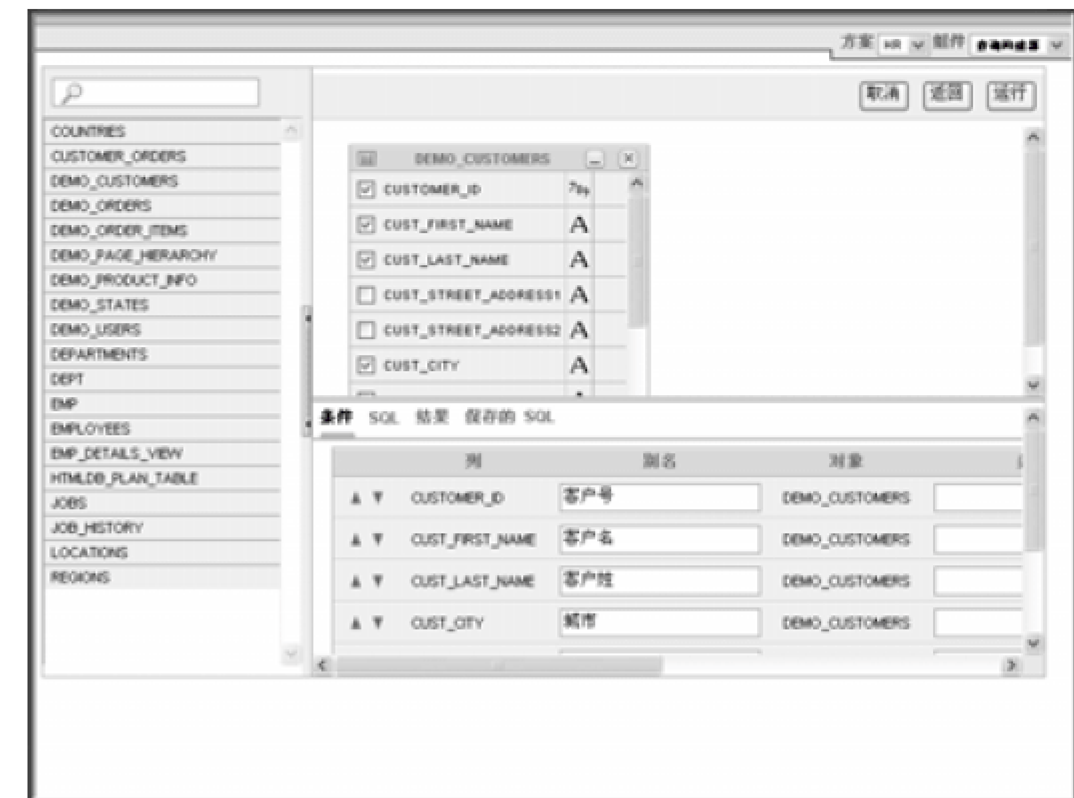


图 22.103

（11）选择 SQL 选项卡将得到通过刚才的操作所创建的 SQL 查询语句，如图 22.104 所示。现在读者可以相信即使没有 SQL 的知识也可以学习 Oracle Application Express 并不是吹牛皮了，因为查询构建器根据用户在图形界面上的操作自动地生成相应的 SQL 查询语句。其实，Express 的查询构建器就是一个代码生成器。

（12）单击“运行页”图标，然后再选择“结果”选项卡，就会得到以上 SQL 语句的运行结果，如图 22.105 所示。

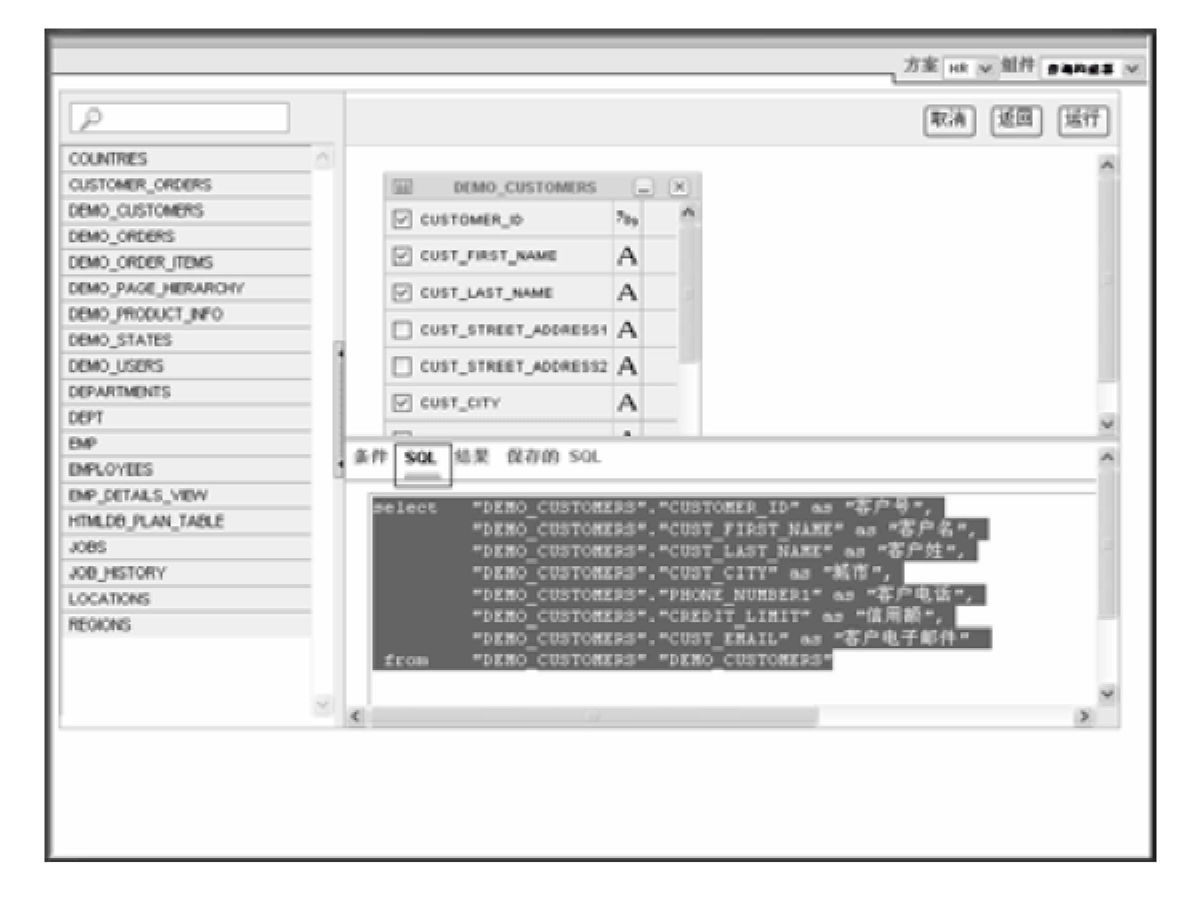


图 22.104

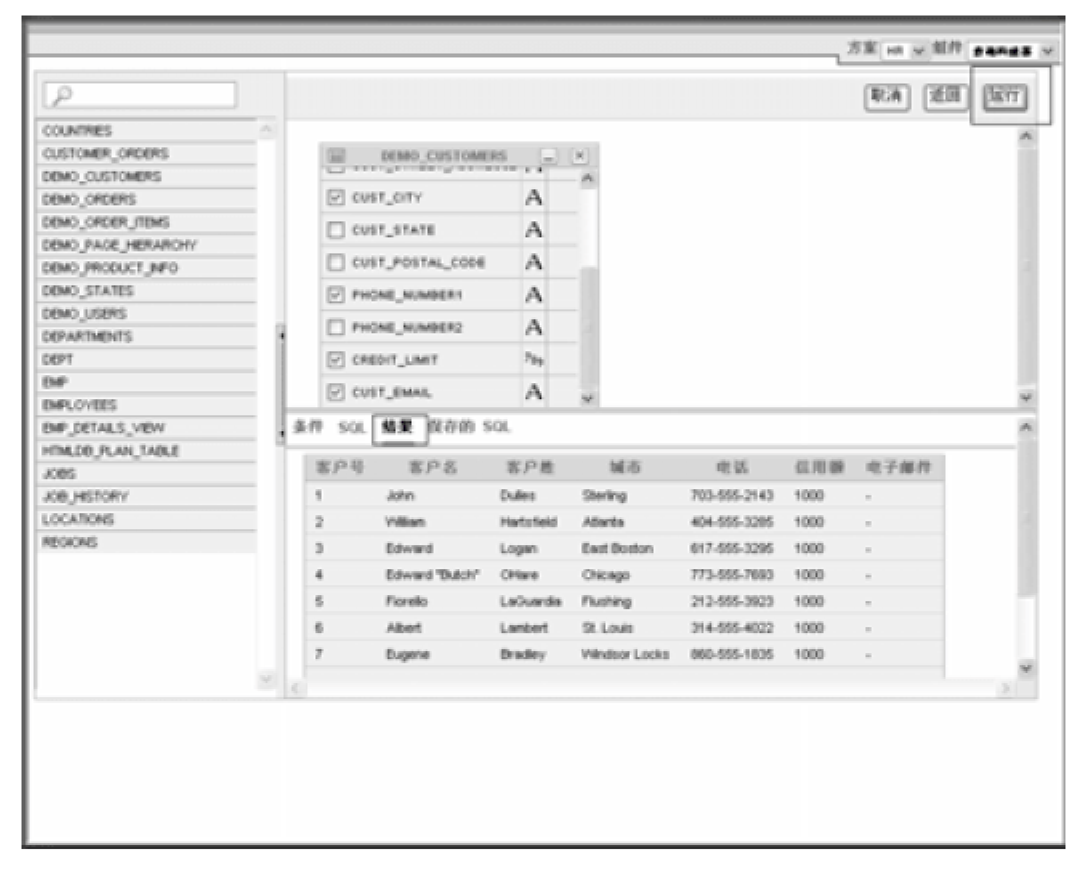


图 22.105

（13）再次选择 SQL 选项卡，然后选择所有的 SQL 语句并复制该 SQL 语句，最后单击“返回”按钮以返回“创建区域”页面，如图 22.106 所示。

（14）将 SQL 语句粘贴到相应的区域后，单击“下一步”按钮，如图 22.107 所示。

（15）全部接受默认值，单击“下一步”按钮，如图 22.108 所示。

（16）全部接受默认值，单击“创建区域”按钮，如图 22.109 所示。之后会返回页 1 的定义页面。

（17）在页 1 的定义页面的“区域”栏的最底部多了一个显示点，单击左上角的“运行页”图标运行应用程序，如图 22.110 所示。

最后就会得到如图 22.111 所示的页面显示。您这么快就基本完成了武大郎烧饼总公司

的互联网应用程序的开发，是不是觉得自己也成为一个编程高手了？



图　22.106



图　22.107

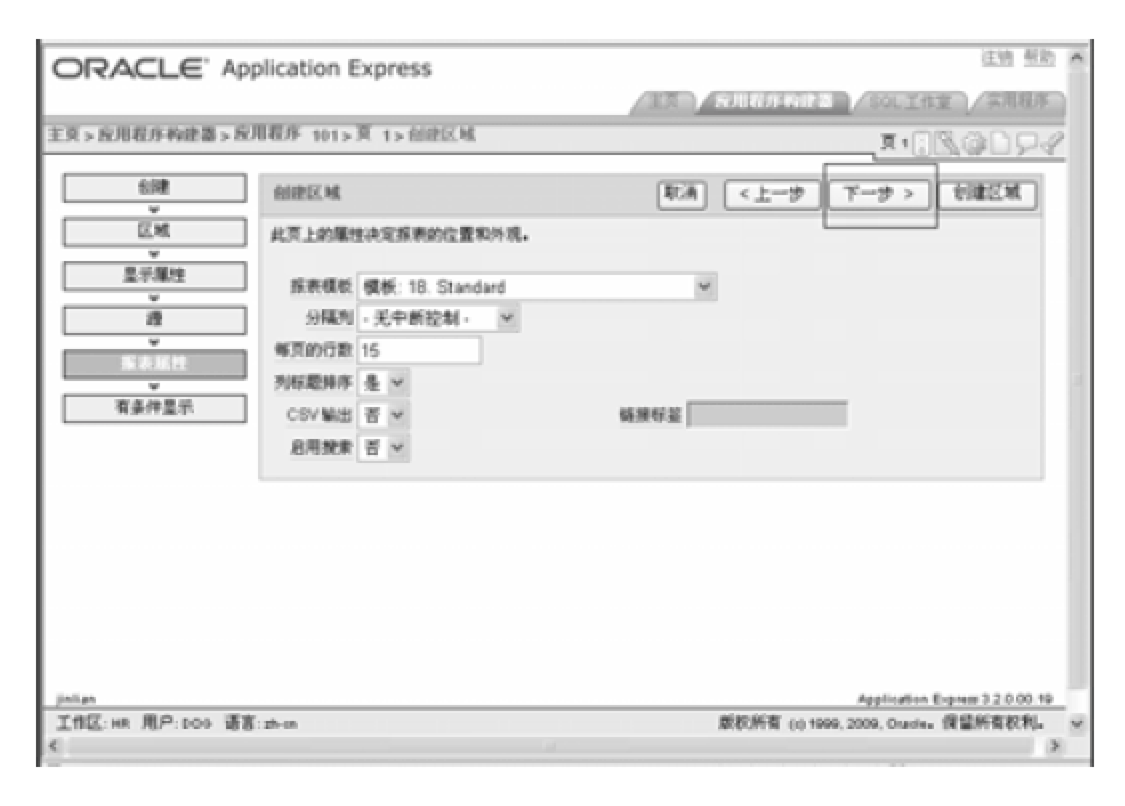


图　22.108

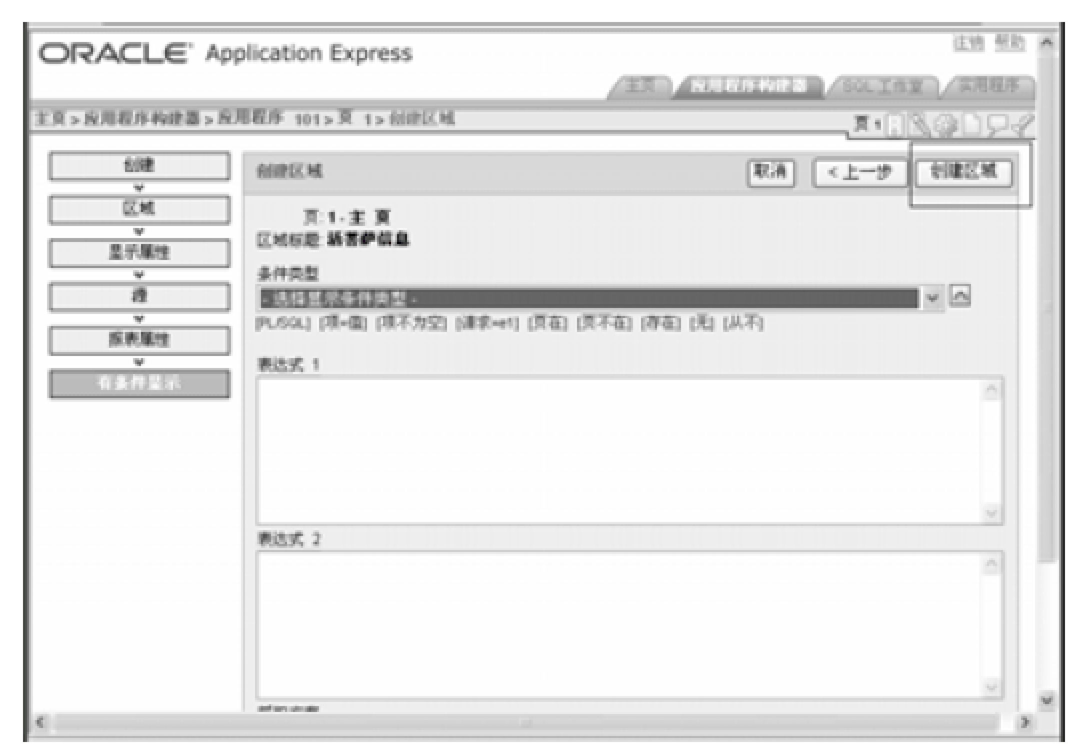


图　22.109



图　22.110

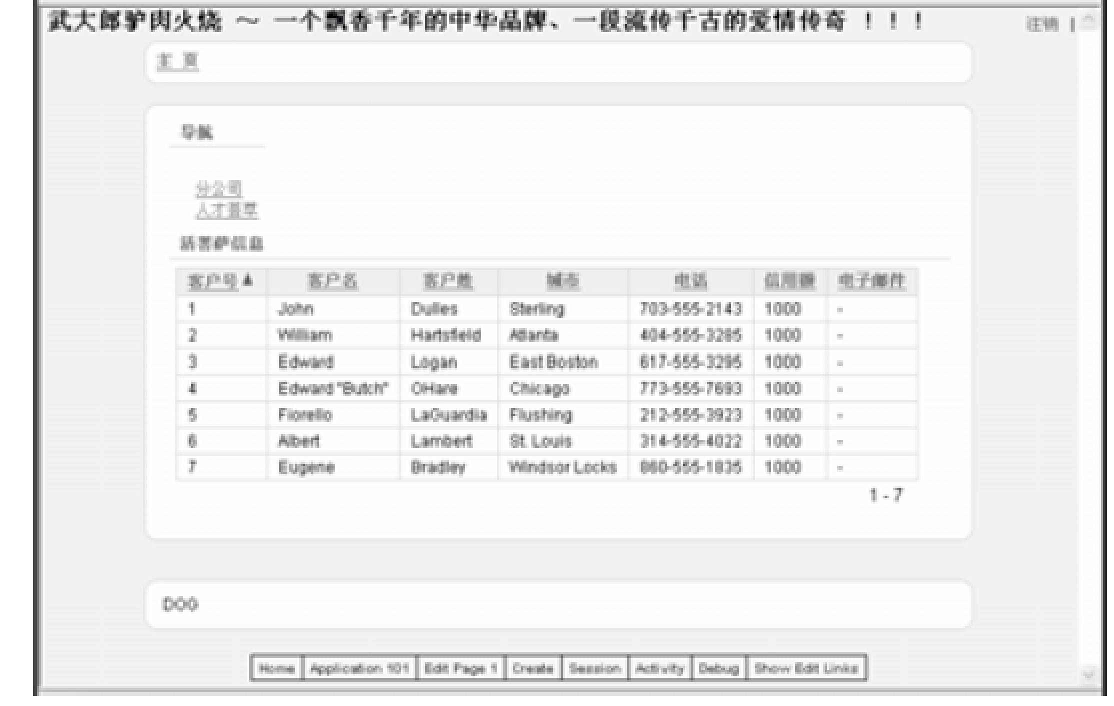


图　22.111

22.12　在网页上添加图形

为了后面的操作方便，我们首先在数据库的 HR 用户中增加一个名为 customer_orders 的视图。该视图包含了用户的基本信息和定货的总量(钱数)，它们分别来自 demo_customers

和 demo_orders。以下就是具体的操作：

（1）启动 DOS 窗口，使用例 22-1 的 SQL*Plus 命令以 HR 用户登录数据库系统。

例 22-1

```
sqlplus hr/hr
```

（2）使用例 22-2 的 DDL 语句创建视图 customer_orders（其中，or replace 表示如果 customer_orders 视图已经存在，就使用新的定义覆盖掉）。

例 22-2

```
create or replace view customer_orders
as
select substr(cust_last_name,1,1)||', '||cust_first_name name,
order_id, order_total
from demo_customers, demo_orders
where demo_customers.customer_id = demo_orders.customer_id;
```

虽然老板对您开发的应用程序很满意，但是她觉得网页只有文字和数字显得不太美观，她要求您将客户的订货量（钱数）以图形方式放在公司的主页上。于是您开始了如下的操作：

（1）在主页的定义页面的“区域”栏中单击“创建”图标，如图 22.112 所示。之后将进入“创建区域”页面。

（2）选中“图表”单选按钮，单击“下一步”按钮，如图 22.113 所示。



图 22.112

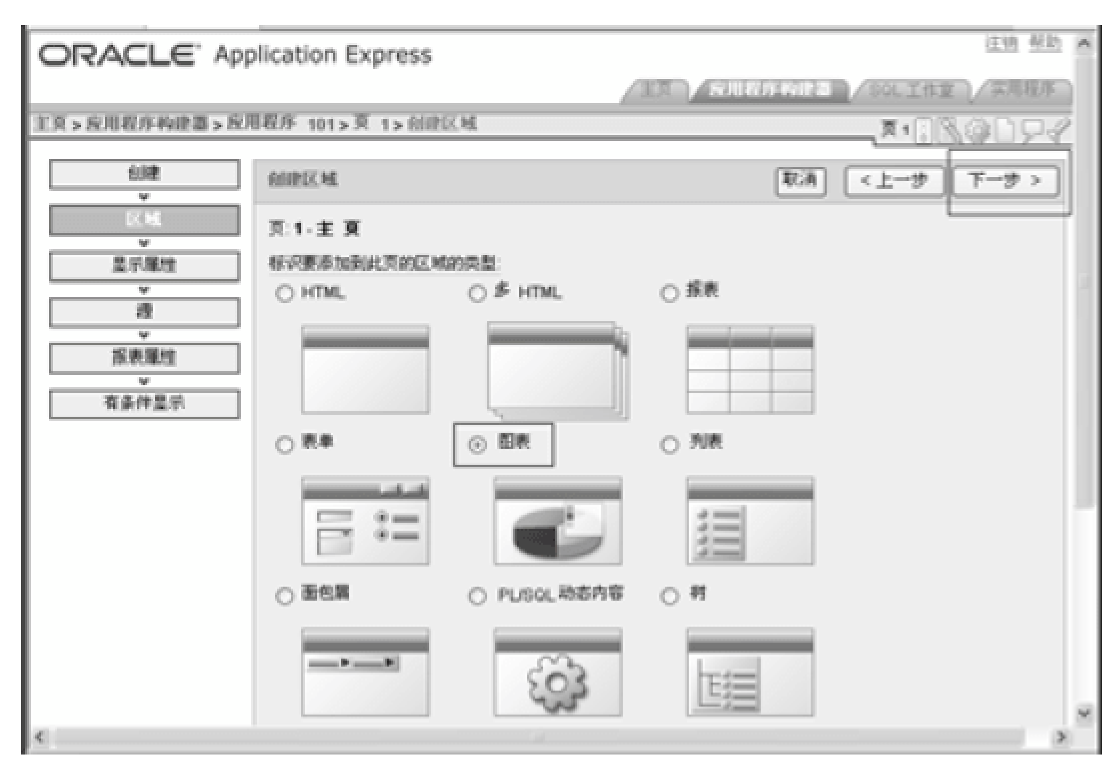


图 22.113

（3）选中“Flash 图表”单选按钮，单击“下一步”按钮，如图 22.114 所示。

（4）在“标题”文本框中输入“活菩萨的贡献”，在“区域模板”下拉列表框中接受默认选择的 Chart Region，在“序列”文本框中输入“20”（这样就可以保证这个图形显示在活菩萨信息之上，因为那个区域的序列为 30），然后单击“显示点”下拉列表框右侧的“发现”图标，如图 22.115 所示。之后将出现“区域显示点”对话框。

（5）单击“区域位置 1”超链接，如图 22.116 所示。之后回到“创建区域”页面。

（6）单击“下一步”按钮，如图 22.117 所示。

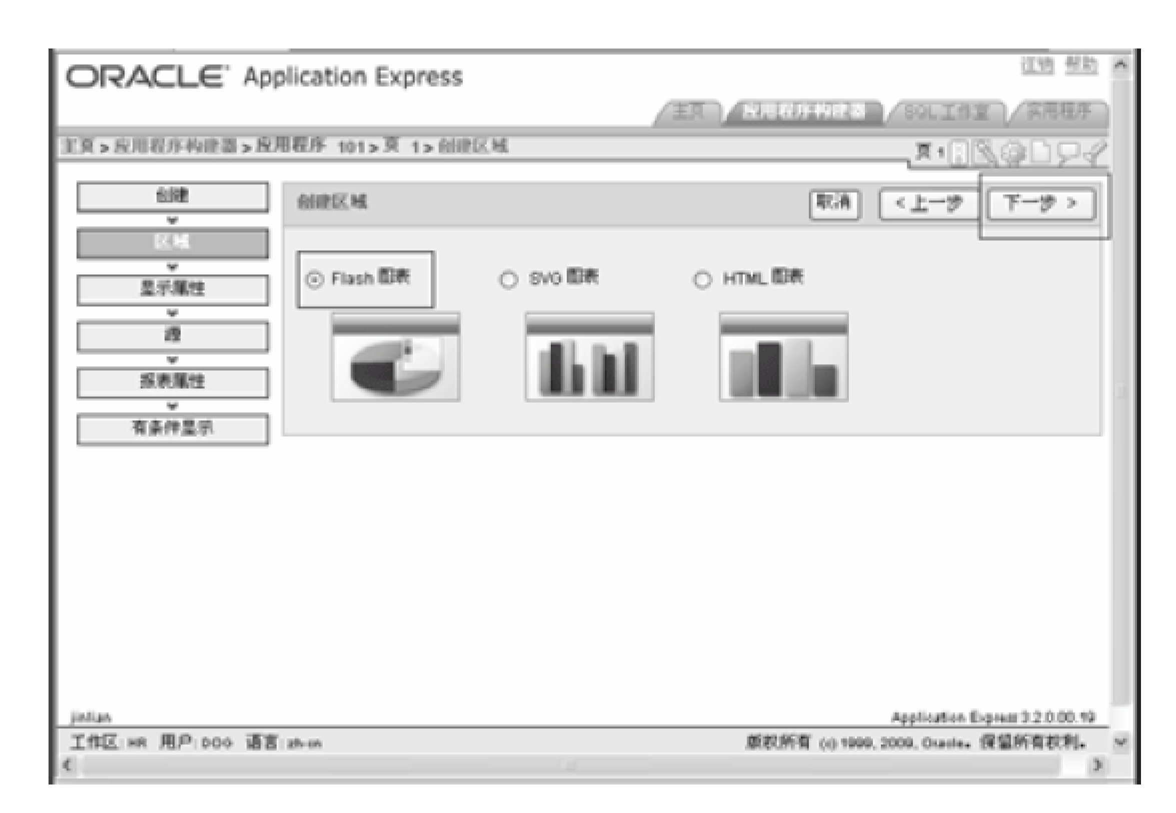


图　22.114



图　22.115

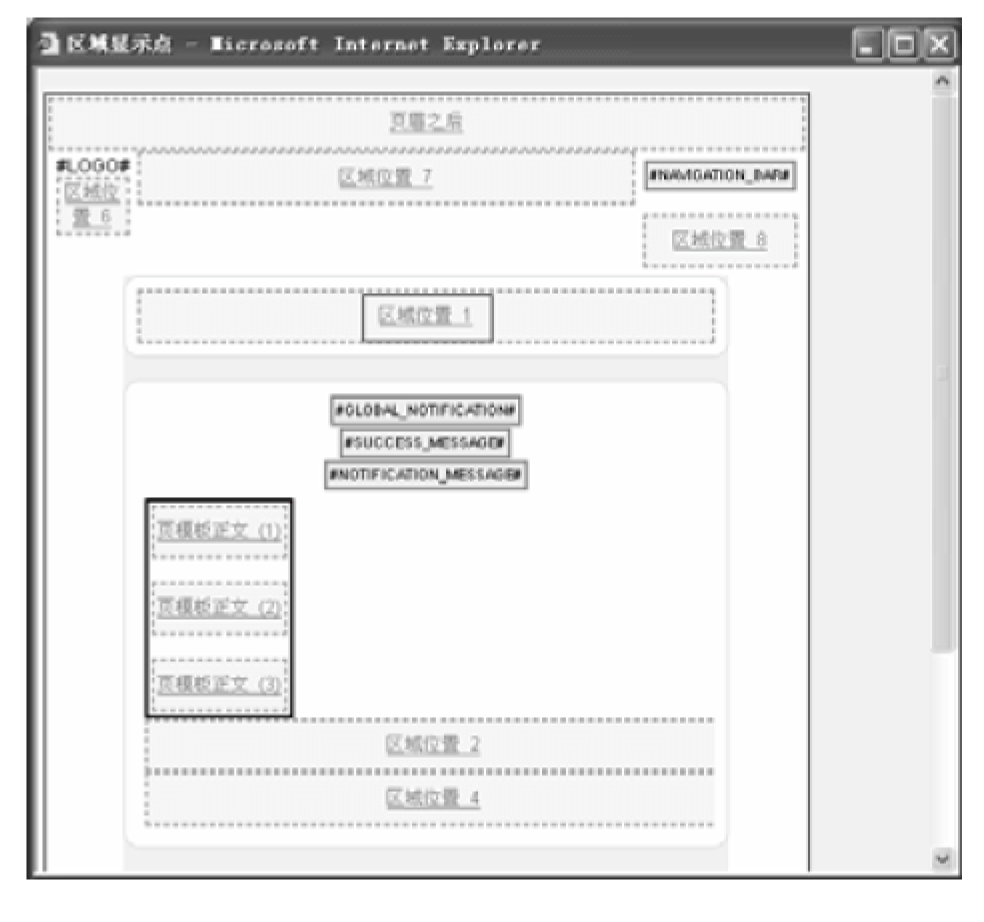


图　22.116



图　22.117

（7）在“图标类型”下拉列表框中选择“3D 饼图”（考虑到与烧饼比较相似），在“图标标题”文本框中输入“姓名”，选中“显示标签”复选框，在“显示图例”栏中选中“有”单选按钮，如图 22.118 所示。

（8）单击“下一步”按钮，如图 22.119 所示。

（9）在“输入 SQL 查询或返回 SQL 查询的 PL/SQL 函数”列表框输入如下的 SQL 查询语句（随书的光盘上的 chart.sql 文件中包含该查询语句，读者可以将其复制到该列表框中）。

```
SELECT NULL LINK,
name LABEL,
sum(order_total) VALUE
FROM customer_orders
GROUP BY name
```

其中，LINK 是一个 URL（网页地址），LABEL 是在饼图中显示的文字，值是定义饼图大小的数字列。之后单击“下一步”按钮，如图 22.120 所示。

（10）接受默认设置，单击“创建区域”按钮，如图 22.121 所示。之后将出现主页的定义页面。

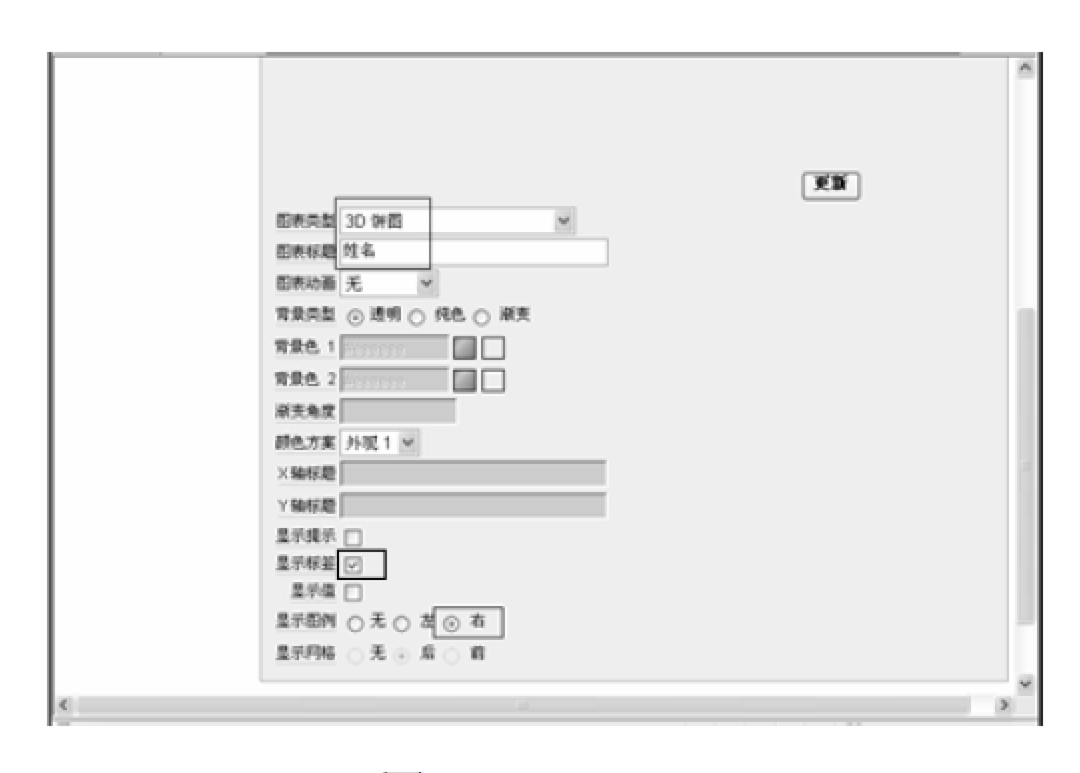


图 22.118



图 22.119



图 22.120



图 22.121

（11）在主页的定义页面的“区域”栏中会出现一个新的区域，其序列为 20，如图 22.122 所示。

（12）单击右上角的“运行页”图标运行该网页，如图 22.123 所示。之后将出现登录界面。



图 22.122



图 22.123

（13）在“用户名”文本框中输入“dog”，在“口令”文本框中输入“xm_Q1ng”，单击“登录”按钮，如图 22.124 所示。之后将显示应用程序的主页。

（14）虽然图表按要求显示了，但看上去并不美观，如图 22.125 所示，于是您退回到主页的定义页面。

（15）为了修改图表，在“区域”栏单击“Flash 图表”超链接，如图 22.126 所示。之后出现“Flash 图表属性”页面。

（16）在“图表类型”下拉列表框中选择“3D 饼图”，在“图表标题”文本框中清除原来定义的“姓名”，将图表的宽度和高度都减少到原来的一半，分别为 300 和 200，如图 22.127 所示。

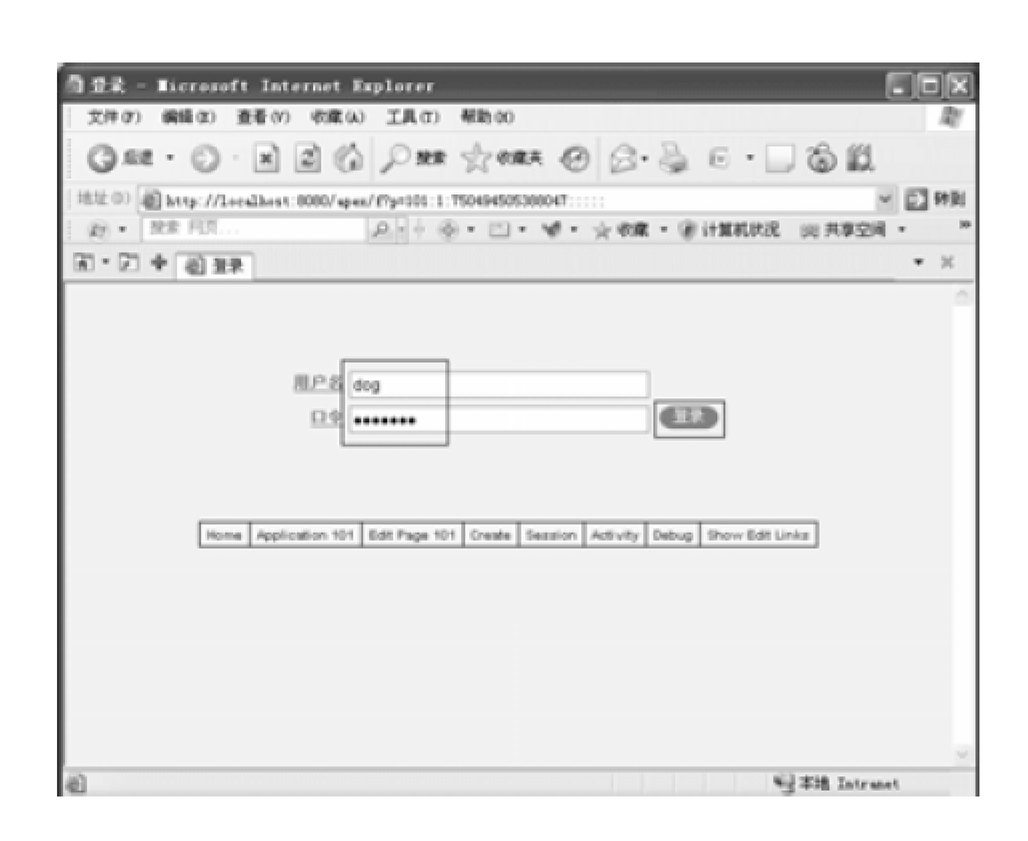


图　22.124

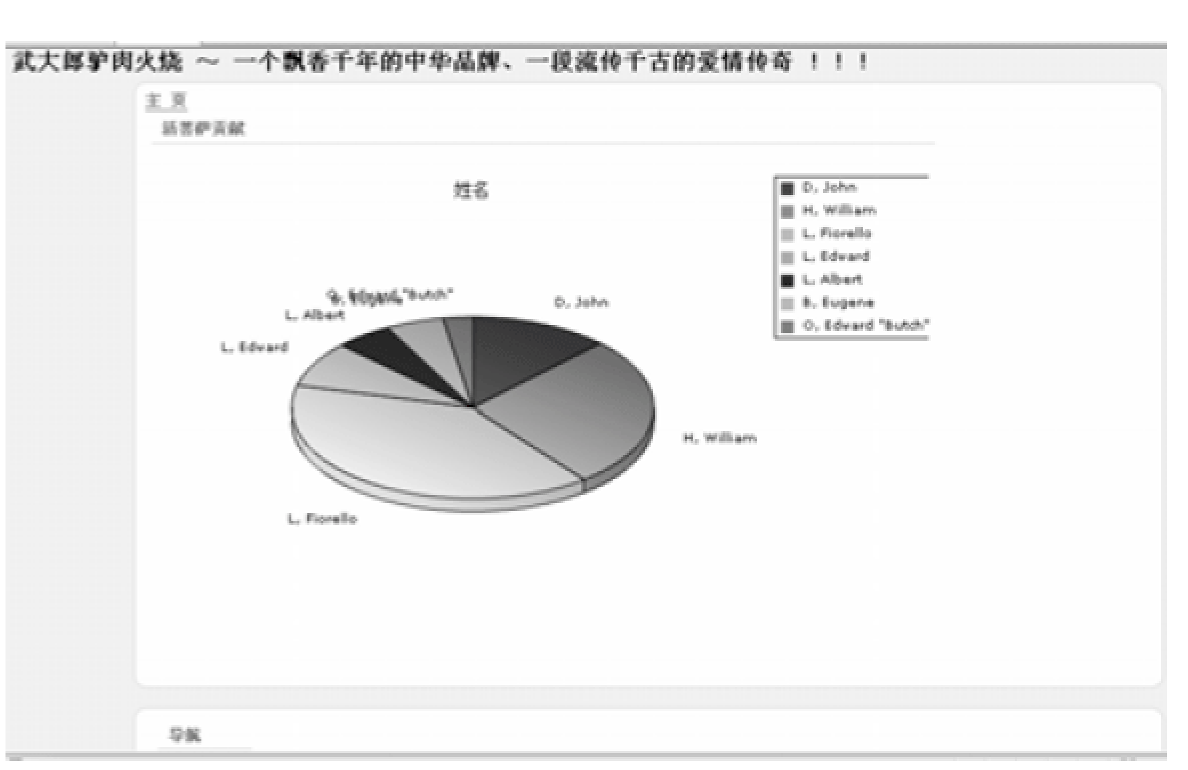


图　22.125



图　22.126



图　22.127

（17）单击“应用更改”按钮，如图 22.128 所示。之后会退回到主页的定义页面。

（18）单击“运行页”图标以运行应用程序，如图 22.129 所示。之后将显示如图 22.130 所示的页面。



图　22.128



图　22.129

（19）尽管图表的显示有了些改善，但是图例好像太宽了。为了缩小图例的宽度，单击最底部的开发工具栏中的 Home 超链接，如图 22.131 所示。



图　22.130



图　22.131

（20）单击“SQL 工作室”图标，如图 22.132 所示。

（21）单击“对象浏览器”图标，如图 22.133 所示。

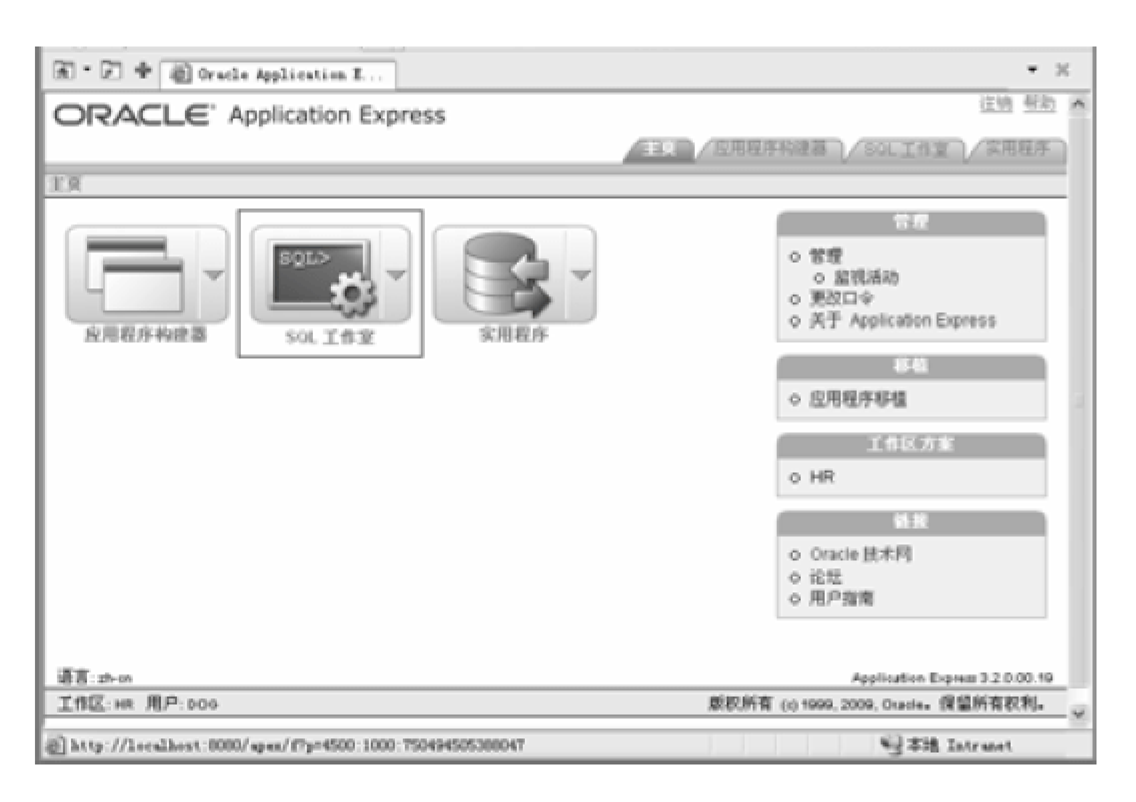


图　22.132



图　22.133

（22）选择 DEMO_CUSTOMERS 表，选择“数据”选项卡，单击第 4 行数据最左边的编辑图标，如图 22.134 所示。

（23）在 Cust First Name 文本框中清除 Butch 后，单击“应用更改”按钮，如图 22.135 所示。



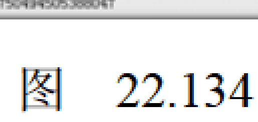


图　22.134



图　22.135

（24）之后将出现如图 22.136 所示的页面，此时退回到页的定义页面。

（25）由于您想将图表移到导航部分，所以在“区域”栏中单击“活菩萨贡献”超链

接，如图 22.137 所示。之后将进入“编辑区域”页面。



图　22.136



图　22.137

（26）在“显示点”下拉列表框中选择“页模板区域位置 2”选项，在“列”下拉列表框中选择 9，单击“应用更改”按钮，如图 22.138 所示。之后将退回到页 1 的定义页面。

（27）单击“运行”按钮运行应用程序，如图 22.139 所示。



图　22.138



图　22.139

（28）通过显示的页面，您觉得活菩萨信息在显示的网页上占的位置太大，因此决定将其显示的信息改为每页只有两行。于是您先退回到主页的定义页面，然后在“区域”栏单击序列为 30 的区域的报表，如图 22.140 所示。

（29）在“报表属性”页面的“行数”文本框中输入“2”，如图 22.141 所示。



图　22.140



图　22.141

（30）单击“应用更改”按钮，如图 22.142 所示。之后将退回到主页的定义页面。



图　22.142

（31）单击“运行页”图标，如图 22.143 所示。



图　22.143

最后您终于得到了一个比较满意的网页显示，如图 22.144 所示。



图　22.144

当您仔细端详了一会儿这个网页之后，总觉得这个画面有似曾相识的感觉。当您突然想起它特别像电影中日本鬼子的膏药旗时，您决定必须要再一次修改网页的显示。正在这

时，老板要立即见您并要求您带上刚开发好的应用程序。现在您只得带上这个像“膏药旗”一样的网页来让她老人家欣赏了。

当老板看了您的“膏药旗”网页之后突然拍了一下桌子，您心想完了，就等着老板教训自己了。可她接下来的话却让您大吃一惊。她说：“太棒了！这正合我意。那个图案看上去整个就是一个武大郎烧饼。仔细端详咱们的烧饼就像悬在半空中的太阳。您的构思和设计堪称完美。”听完老板的话，您也明白了。在老板心目中，咱们一衣带水的友好邻邦的国旗就是白布包皮上钉了一个武大郎的烧饼。

老板一高兴接着说：“前几天，一个海归（她的一个亲戚的孩子）送给了她一个从国外带来的艺术品，叫什么丝，一个没胳膊的女人雕像，据说是一位有名的美女。这留了几年洋，脑子都灌了驴肉汤了，我这是做生意开店的，店里供个缺胳膊少腿的女人也不是什么好兆头啊！你看看，这西洋鬼子的头真是都被驴给踢了。现在这电视上整天选小姐（选美），从来就没见到一个缺胳膊少腿的女人参选。这缺胳膊少腿的女人别说选小姐选不上，就是我这烧饼店招伙计也不敢要。再看咱这烧饼，那天生就是一个艺术精品，随便放在什么上都是一副美轮美奂的画卷。”

第23章 管理数据和部署应用程序

在本章中，首先介绍如何使用 Oracle Application Express 提供的图形工具方便地完成 Oracle 与其他系统之间数据格式的转换，之后将介绍在开发环境中如何将开发的应用程序部署到生产环境中。

23.1 数据加载/卸载工具（数据车间）

要使用 Oracle Application Express（快速 Web 应用开发工具）创建数据库驱动的应用程序，应用程序所需的数据必须存储在 Oracle 数据库中。那么如何将数据装入 Oracle 数据库，又如何从数据库中抽取数据呢？Oracle Application Express 中的数据加载/卸载工具（在 Oracle 10g 中称为数据车间）就是用来完成这一重任的。图 23.1 为数据加载/卸载工具的工作原理示意图。

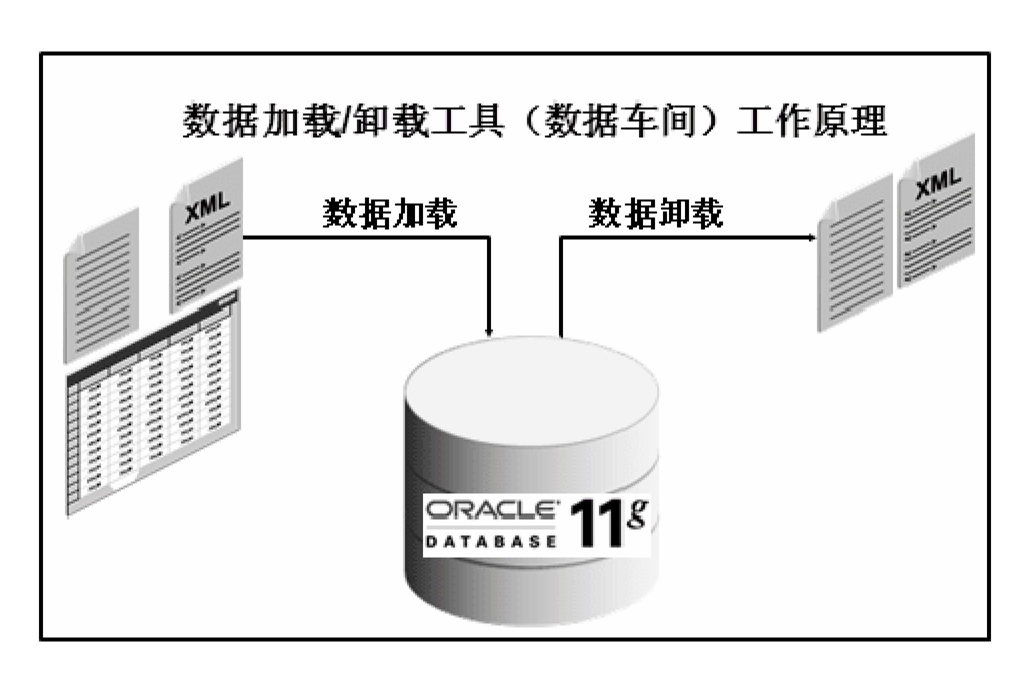


图 23.1

当您要装入较少的数据以设置数据库并开始创建一个应用程序时，数据加载/卸载工具特别有用。这一工具将呈现的信息存储到 Oracle 数据库中，而且数据的格式可以为以下形式的任何一种：

- 纯文本文件，数据由分隔符所分割。其分隔符可以是逗号、空格或制表键等。
- 电子表格文件。
- 可延伸（可扩充）标记语言（XML）文档。

因此，通过使用数据加载/卸载工具从电子表格、XML 文档、其他纯文本文件中抽取数据，开发人员可以快速地创建数据库表。

需要指出的是：使用数据加载/卸载工具并不需要加载所有的数据。当数据量较小时它

是一个理想的加载工具，但是如果要加载大量的数据时，还是要使用如 Oracle SQL*Loader 或外表之类的专门数据加载工具。

另外，因为 Oracle Application Express 是建在 Oracle 数据库之中的，所以任何已经存储在 Oracle 数据库中的数据，无论存在哪个用户中，Oracle Application Express 都可以访问。

要访问数据加载/卸载工具，首先要启动网络浏览器并登录到 Oracle Application Express 的主页，然后进行如下的操作：

（1）单击“实用程序”图标，如图 23.2 所示。

（2）单击“数据加载/卸载”图标，如图 23.3 所示。



图　23.2



图　23.3

之后就会进入如图 23.4 所示的页面，现在您就可以加载数据到 Oracle 数据库中或从 Oracle 数据库中卸载数据了。



图　23.4

23.2　将数据卸载到正文文件中

为了讲解方便，我们先介绍如何利用 Oracle Application Express 提供的图形工具将 Oracle 数据库中的数据卸载到一个正文文件中，具体操作步骤如下：

（1）在硬盘上创建一个存放卸载数据的目录（文件夹），这里创建的文件夹为 F:\Express_

Data，如图 23.5 所示。

（2）进入“数据加载/卸载”页面，单击“卸载”图标，如图 23.6 所示。之后将会出现“卸载”页面。



图 23.5



图 23.6

（3）单击“卸载为文本”图标，如图 23.7 所示。之后将会出现“卸载为文本”页面。

（4）“方案”接受默认的 HR，单击“下一步”按钮，如图 23.8 所示。



图 23.7

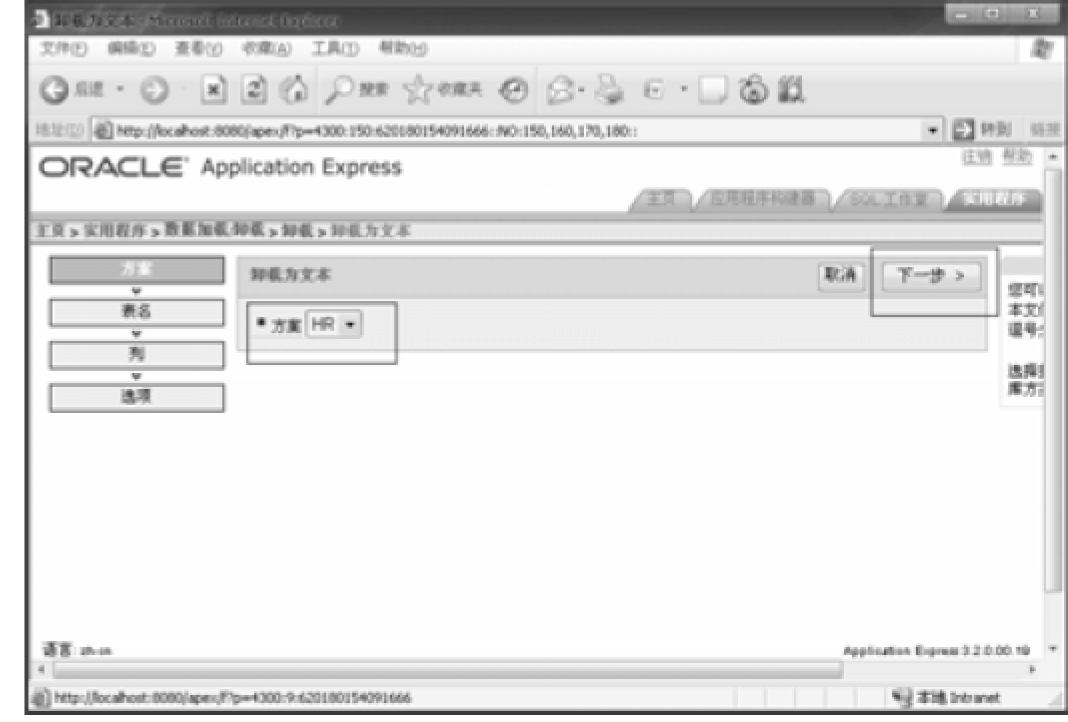


图 23.8

（5）在“表”下拉列表框中选择 DEPT 选项，之后单击“下一步”按钮，如图 23.9 所示。

（6）在“列”列表框中选择所有的列，之后单击“下一步”按钮，如图 23.10 所示。

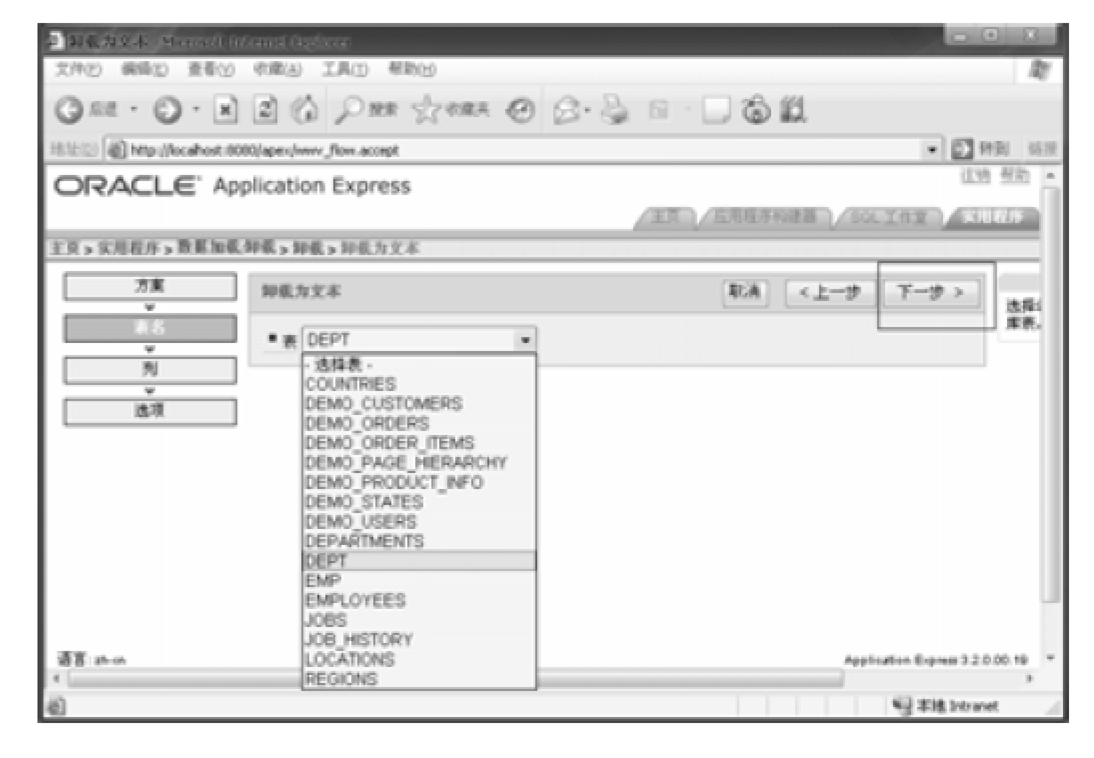


图 23.9

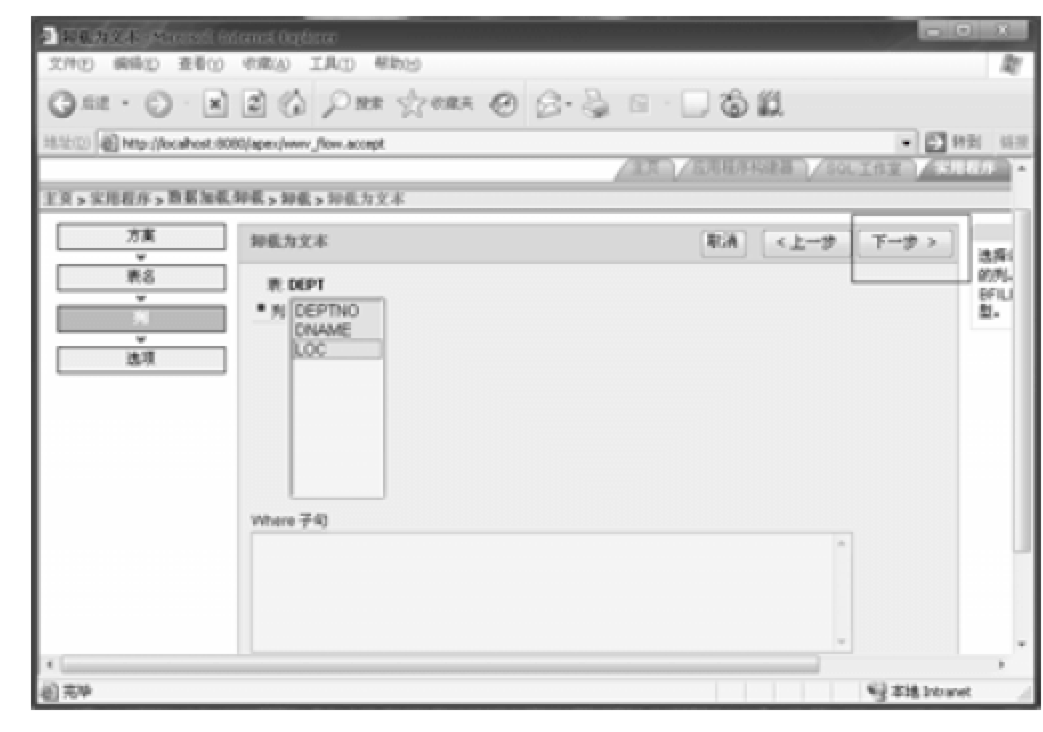


图 23.10

（7）全部接受默认设置，单击“卸载数据”按钮，如图 23.11 所示。之后会出现“文件下载”对话框。

（8）在“文件下载”对话框中单击“保存”按钮，如图 23.12 所示。之后会出现“另存为”对话框。

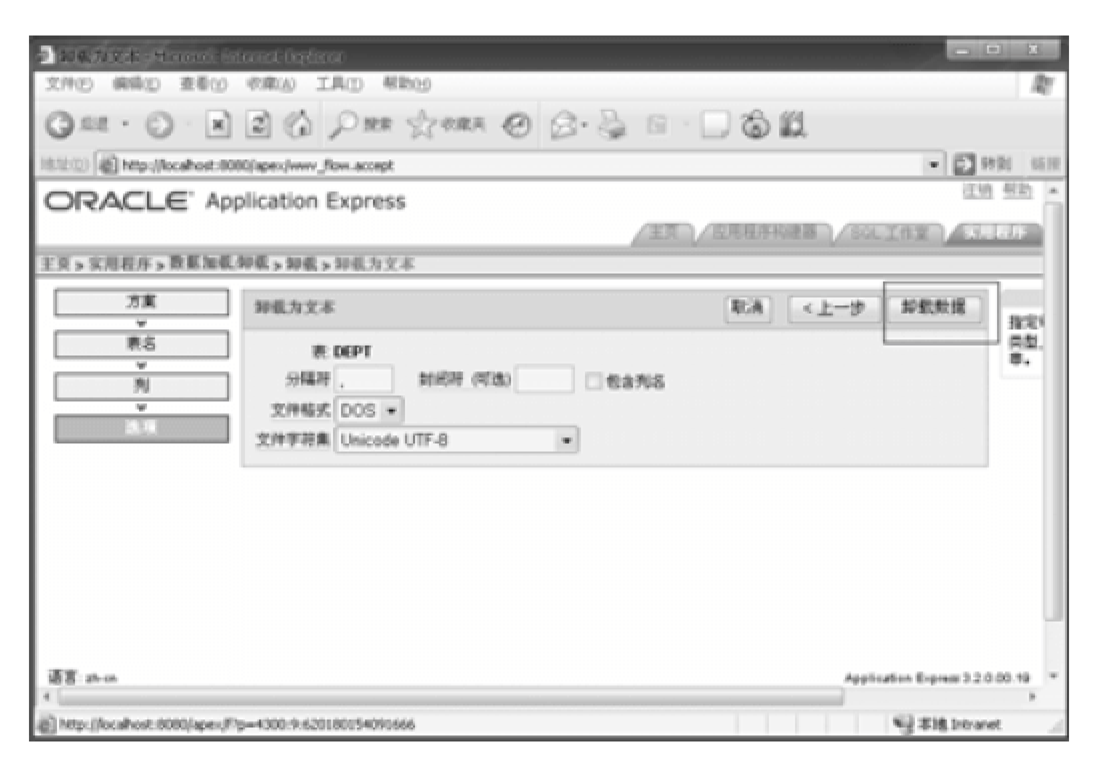


图　23.11



图　23.12

（9）在“保存在”下拉列表框中选择刚创建的 F 盘上的 Express_Data 目录（文件夹），在“文件名”文本框中输入“dept.txt”，单击“保存”按钮，如图 23.13 所示。

（10）进入 Express_Data 目录，双击 dept.txt 文件以打开，如图 23.14 所示。



图　23.13



图　23.14

最后您将看到 dept.txt 文件中的数据，如图 23.15 所示。其他程序设计语言，如 C 可以读取该文件中的数据。原来将 Oracle 数据库中的数据导到其他的系统中就这么简单。现在如果面试时有人问您会不会将 Oracle 的数据导给其他系统，您应该自信地回答“没问题”。



图　23.15

由于可能一些读者没有学习过 XML 语言，所以在这里就不介绍如何将 Oracle 的数据卸载为 XML 的数据了。如果读者有 XML 语言的知识可以自己试着卸载，其操作步骤与这一节所介绍的大致相同。

23.3 将数据卸载到电子表格文件中

将数据卸载到电子表格文件中的操作步骤如下：

（1）首先退回到应用程序的主页，单击“SQL 工作室”图标，如图 23.16 所示。之后将进入“SQL 工作室”页面。

（2）单击“对象浏览器”图标，如图 23.17 所示。之后将进入“对象浏览器”页面。

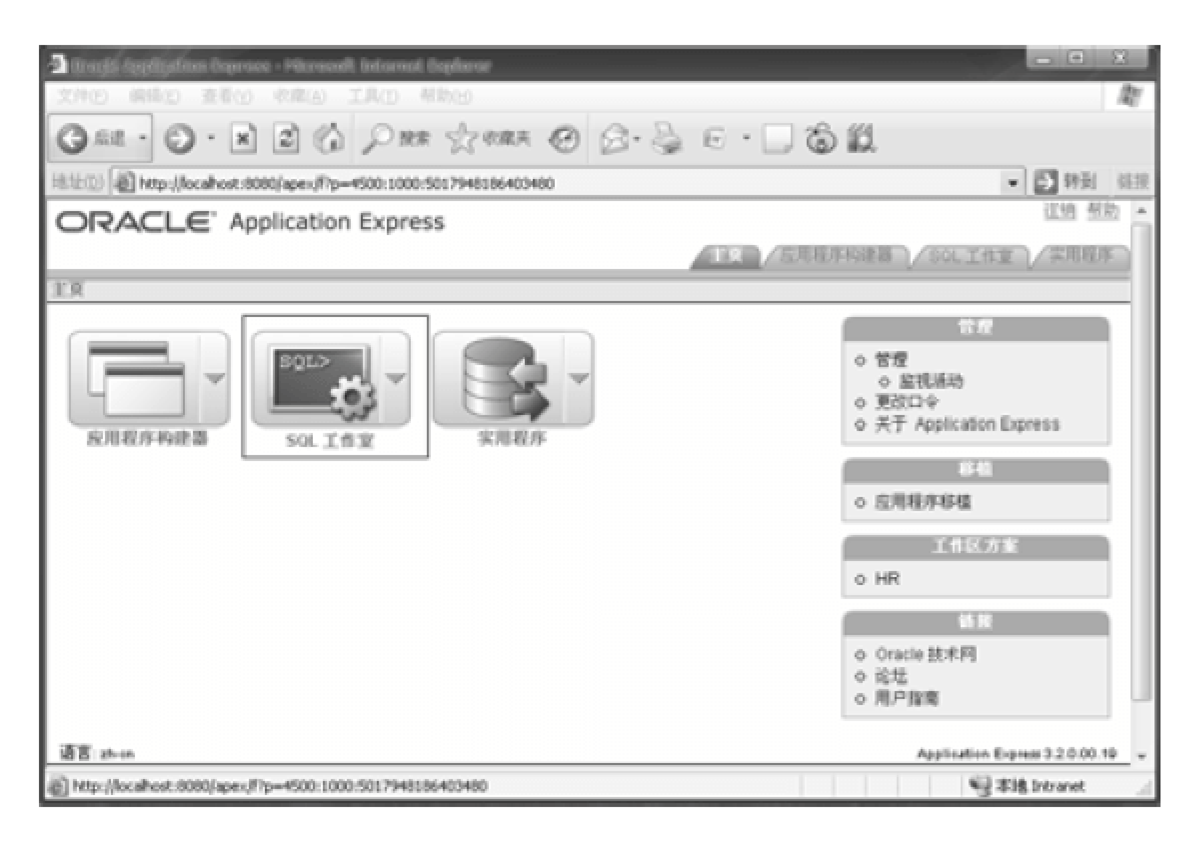


图 23.16

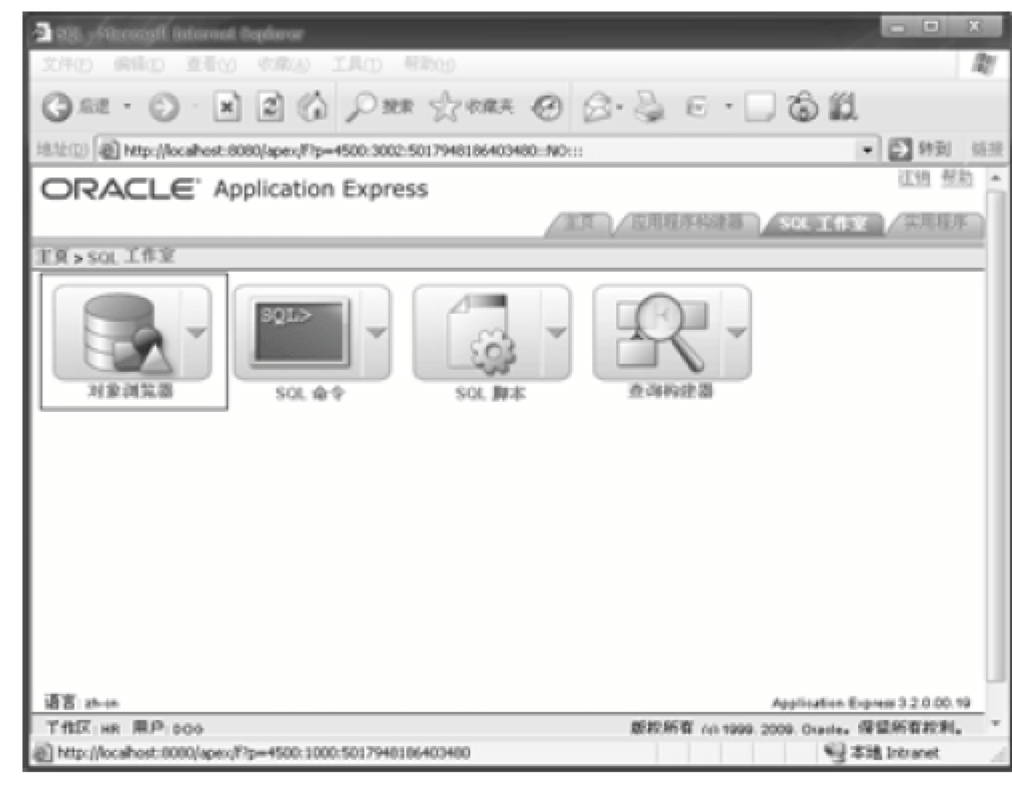


图 23.17

（3）选择要卸载的表（如 EMP），选择“数据”选项卡以查看数据，如图 23.18 所示。

（4）向下滚动滚动条，单击“下载”超链接，如图 23.19 所示。之后将出现“文件下载”对话框。

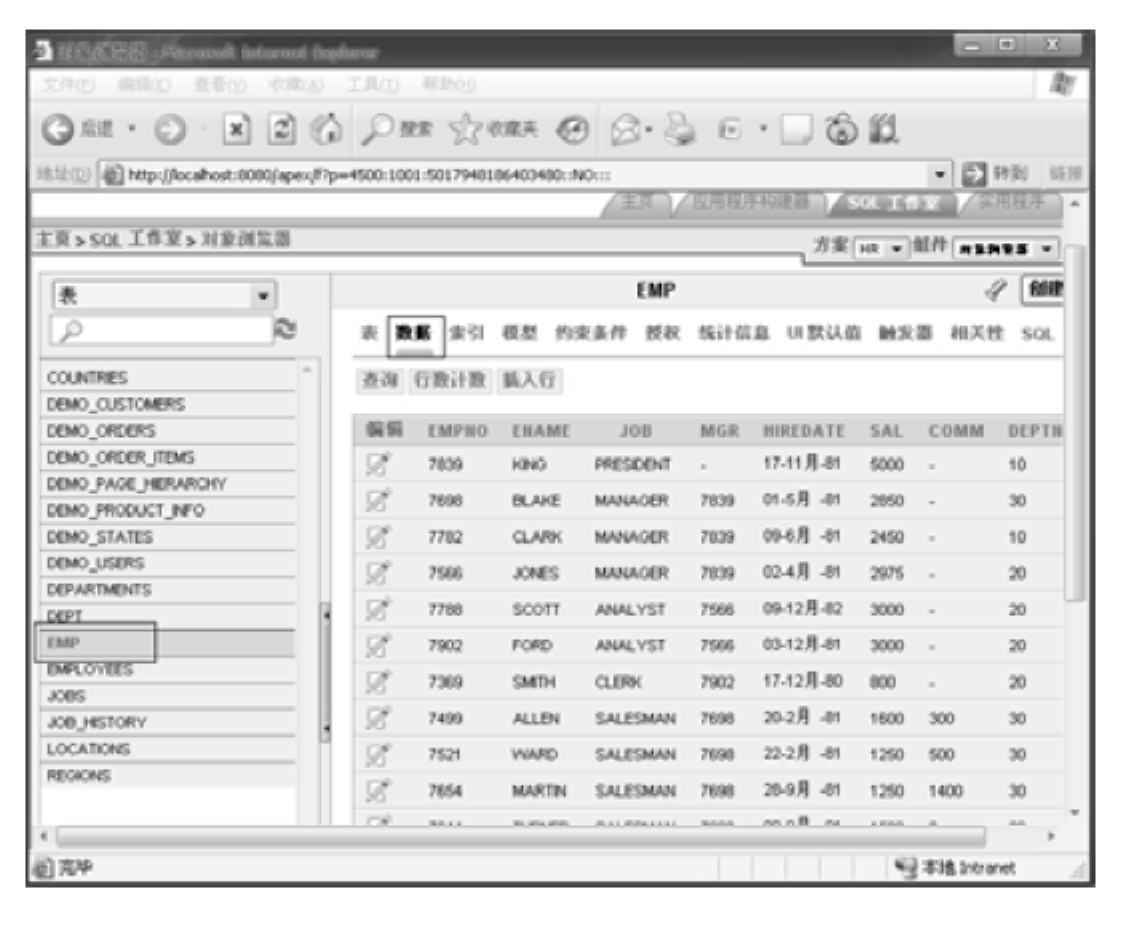


图 23.18

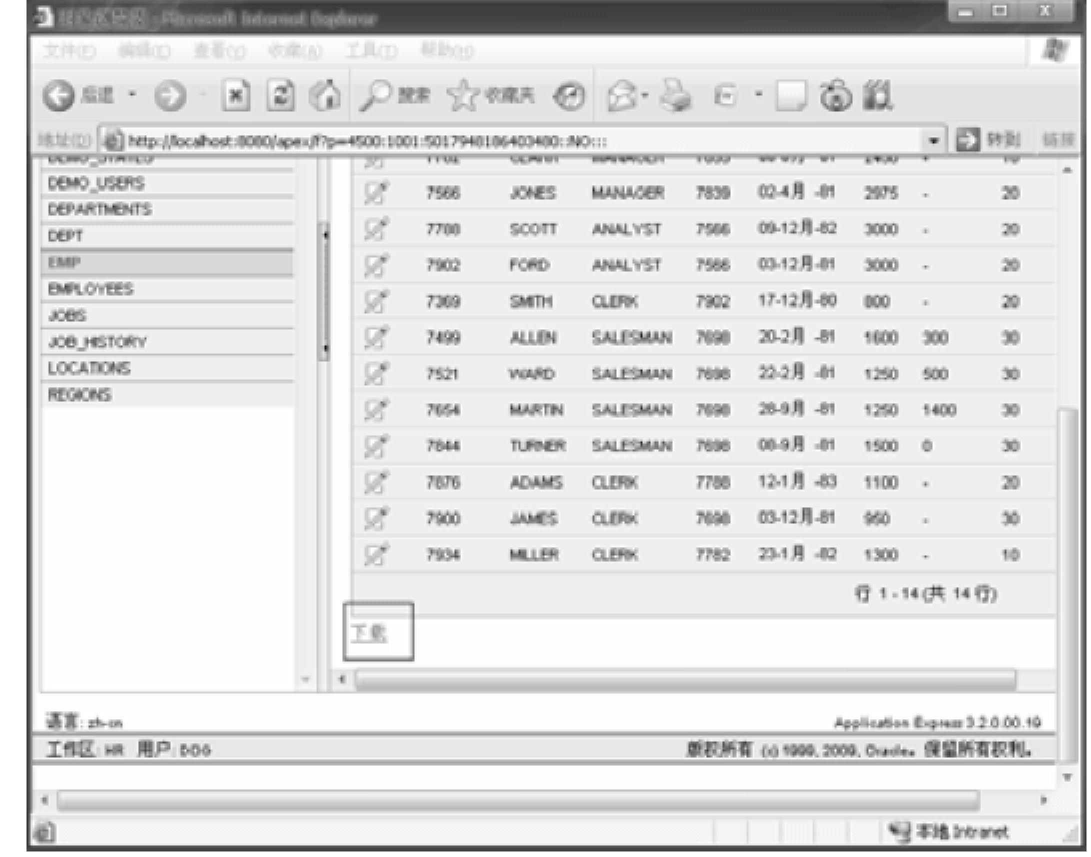


图 23.19

（5）单击“保存”按钮，如图 23.20 所示。之后将出现“另存为”对话框。

（6）在“保存在”下拉列表框中选择 F 盘上的 Express_Data 作为存放电子表格文件的目录，“文件名”和“文件类型”接受默认设置，单击“保存”按钮，如图 23.21 所示。



图　23.20



图　23.21

（7）使用资源管理器进入 Express_Data 目录就可以发现所需的电子表格文件 EMP.csv 已经生成，如图 23.22 所示。

（8）用鼠标左键双击 EMP.csv 文件将打开该电子表格文件，如图 23.23 所示。

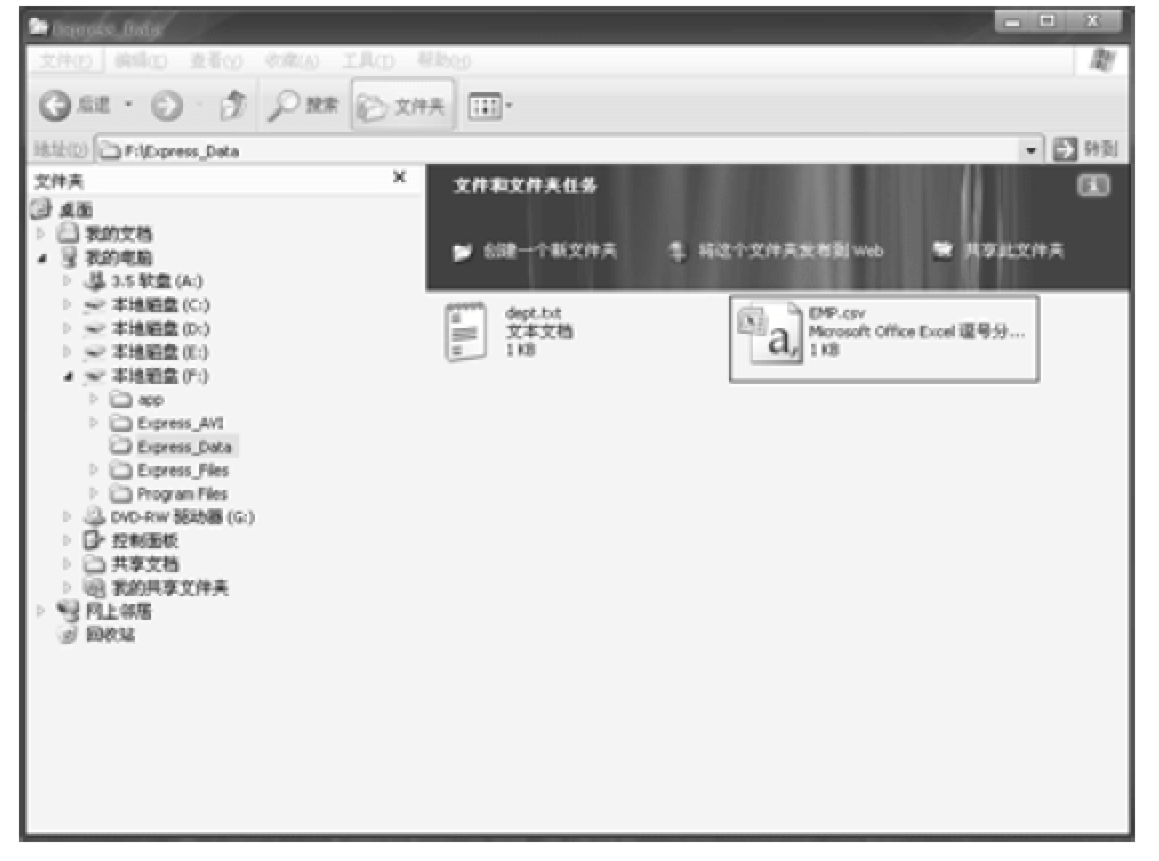
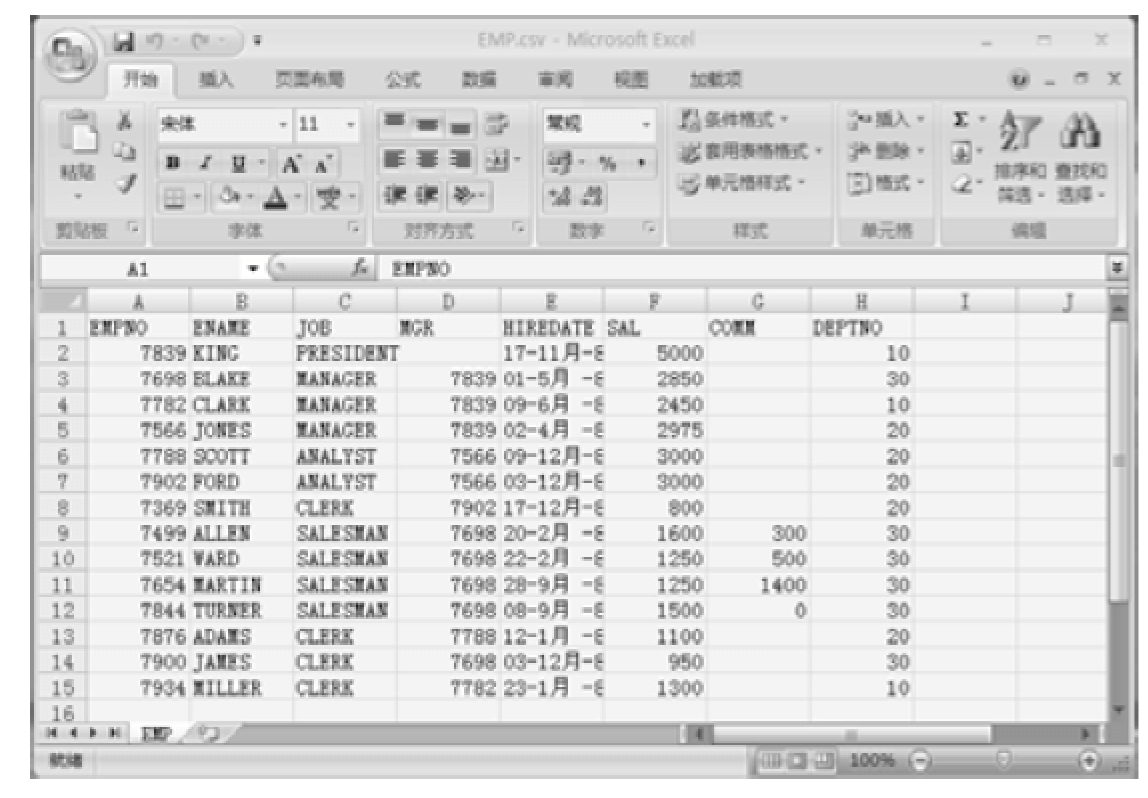


图　23.22



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-11月-8	5000		10
7698	BLAKE	MANAGER	7839	01-5月 -8	2850		30
7782	CLARK	MANAGER	7839	09-6月 -8	2450		10
7566	JONES	MANAGER	7839	02-4月 -8	2975		20
7788	SCOTT	ANALYST	7566	09-12月-8	3000		20
7902	FORD	ANALYST	7566	03-12月-8	3000		20
7369	SMITH	CLERK	7902	17-12月-8	800		20
7499	ALLEN	SALESMAN	7698	20-2月 -8	1600	300	30
7521	WARD	SALESMAN	7698	22-2月 -8	1250	500	30
7654	MARTIN	SALESMAN	7698	28-9月 -8	1250	1400	30
7844	TURNER	SALESMAN	7698	08-9月 -8	1500	0	30
7876	ADAMS	CLERK	7788	12-1月 -8	1100		20
7900	JAMES	CLERK	7698	03-12月-8	950		30
7934	MILLER	CLERK	7782	23-1月 -8	1300		10

图　23.23

之后，那些商业分析员或商业智能人员就可以利用商业智能软件（如 Excel 或 Clementine）对数据进行分析并对市场作出科学的预测。利用商业智能软件所做的预测并不一定总是正确的，没准会像华尔街的金融家们一样分析出一个“金融海啸”来。

23.4　将正文文件的数据加载到Oracle数据库中

在加载数据之前，可能要在数据文件 dept.txt 的最上面（第一行）加上列名，如图 23.24 所示。因为 Oracle Application Express 的加载工具在上传文件时会将文件的第 1 行自动转换成列名。

之后存盘退出。接下来就可以开始将这个纯文本文件的数据上传到 Oracle 数据库中了。具体操作步骤如下：

（1）进入到“数据加载/卸载”页面，单击“加载”图标，如图 23.25 所示。之后将进入“加载”页面。



图 23.24

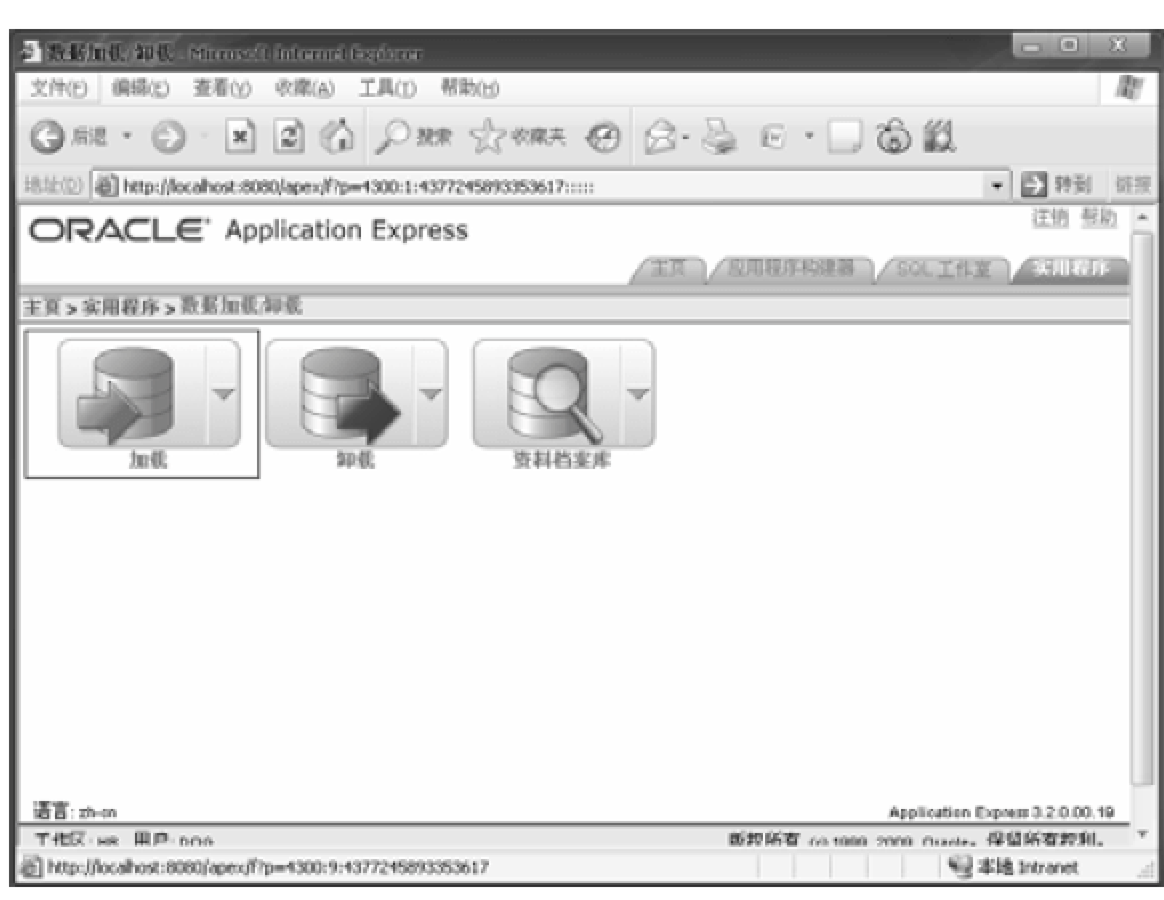


图 23.25

（2）单击“加载文本数据”图标，如图 23.26 所示。之后将进入“加载数据”页面。

（3）选中“新表”和“上传文件”单选按钮，单击“下一步”按钮，如图 23.27 所示。



图 23.26

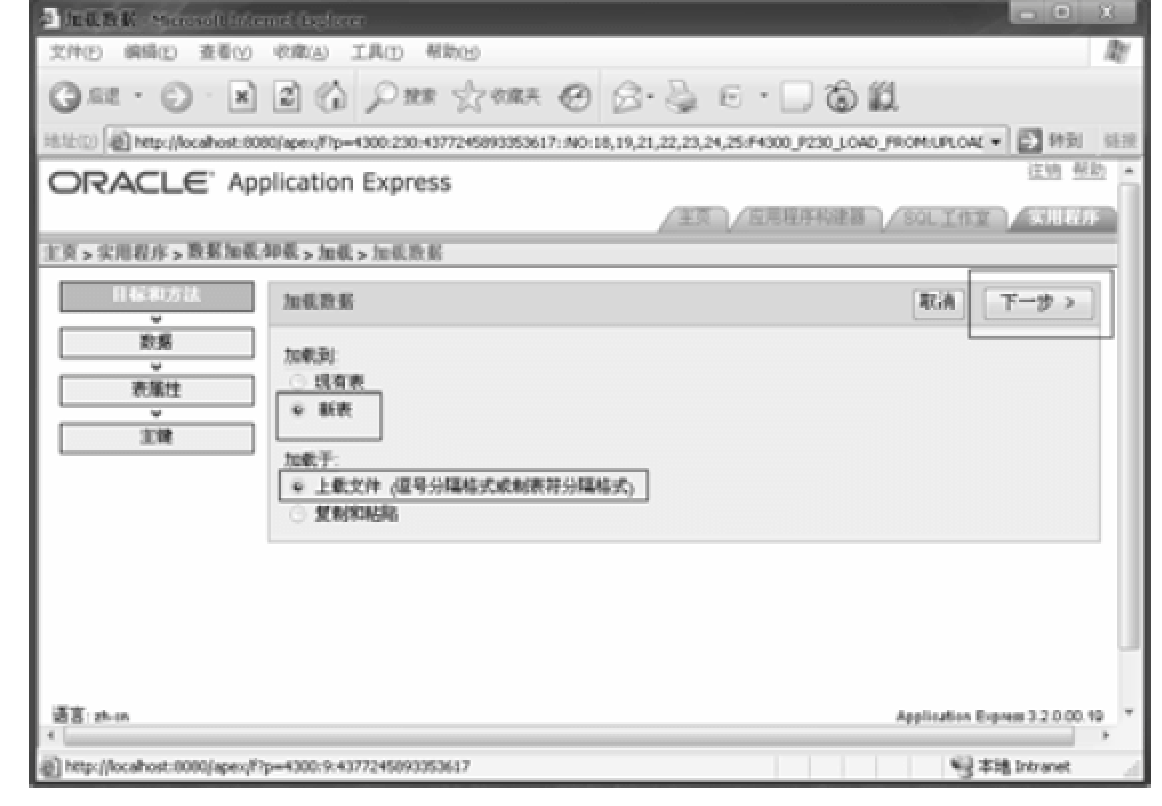


图 23.27

（4）单击“浏览”按钮，如图 23.28 所示。之后将出现“选择文件”页面。

（5）选择 Express_Data 目录下的 dept.txt 文件，单击“打开”按钮，如图 23.29 所示。之后在“文本文件”文本框中将出现 F:\Express_Data\dept.txt（在您的系统上可能是不同的盘符）。

（6）选中“第一行包含列名。”复选框，单击“下一步”按钮，如图 23.30 所示。

（7）在“表名”文本框中输入您认为合适的表名（这里为了方便输入 DEPT2），将 deptno 列的长度修改为 3（也可以修改其他属性），单击“下一步”按钮，如图 23.31 所示。



图　23.28



图　23.29



图　23.30

图　23.31

（8）在“主键来自”栏中选中“使用现有列”单选按钮，在“主键”下拉列表框中选择 DEPTNO(NUMBER)选项，其他选项接受默认设置，单击“加载数据”按钮，如图 23.32 所示。之后将进入“文本数据加载资料档案库”页面。

（9）在“资料档案库”区域将出现刚加载的表 DEPT2 的相关信息，如图 23.33 所示。转到“对象浏览器”页面。



图　23.32



图　23.33

（10）选择“表”，选择 DEPT2，单击“数据”超链接，就可以看到刚上传的数据，如图 23.34 所示。



图 23.34

至此，您已经成功地将一个正文文件中的全部数据上传到 Oracle 数据库并存入到 HR 用户的名为 DEPT2 的表中。

23.5 将电子表格的数据加载到Oracle数据库中

介绍完了将正文数据上传到 Oracle 数据库中的方法之后，接下来我们介绍如何将电子表格数据上传到 Oracle 数据库中，具体操作步骤如下：

（1）进入“数据加载/卸载”页面，单击“加载”图标，如图 23.35 所示。之后将进入“加载”页面。

（2）单击“加载电子表格数据”图标，如图 23.36 所示。之后将进入“加载数据”页面。

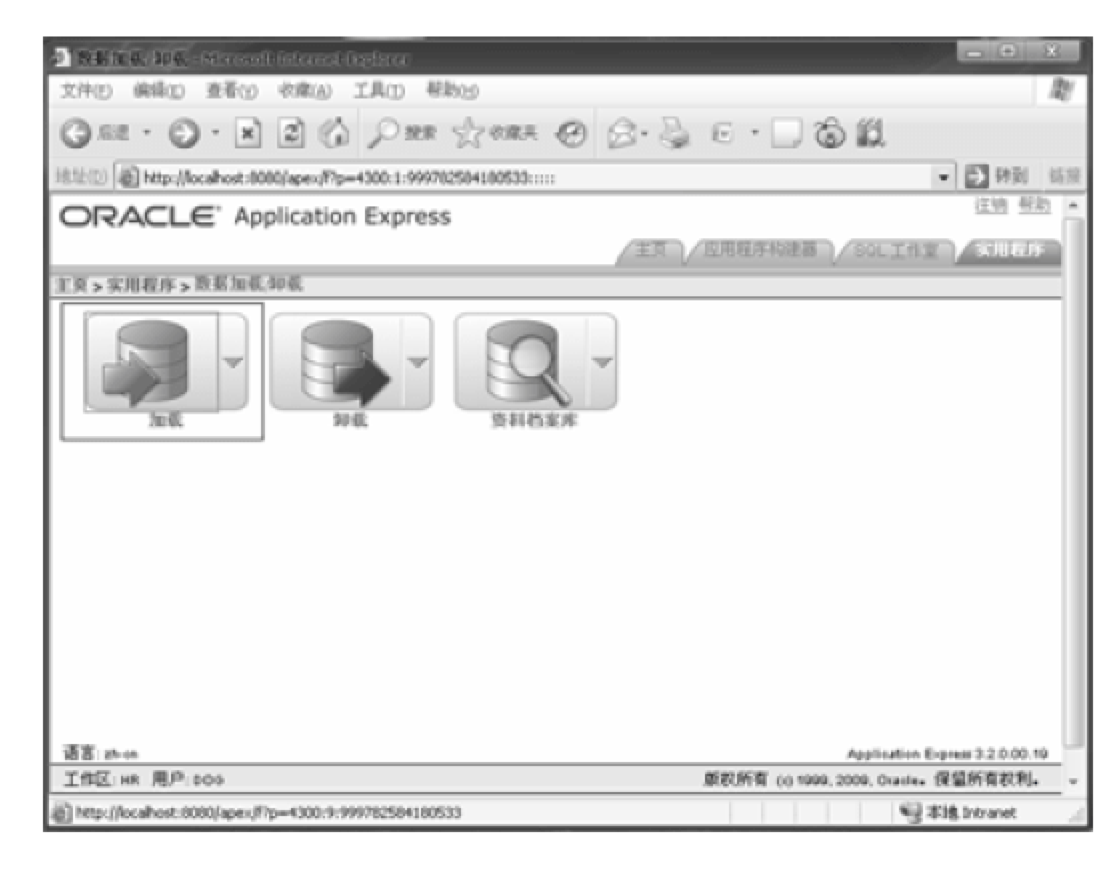


图 23.35



图 23.36

（3）选中“新表”和“复制和粘贴”单选按钮，单击“下一步”按钮，如图 23.37 所示。之后将出现如图 23.38 所示的页面。

（4）打开名为 EMP.csv 的电子表格文件，全选所有的内容，单击“复制”图标，如图 23.39 所示。

（5）将数据粘贴到“数据”列表框中，选中“第一行包含列名。”复选框，单击“下一步”按钮，如图 23.40 所示。

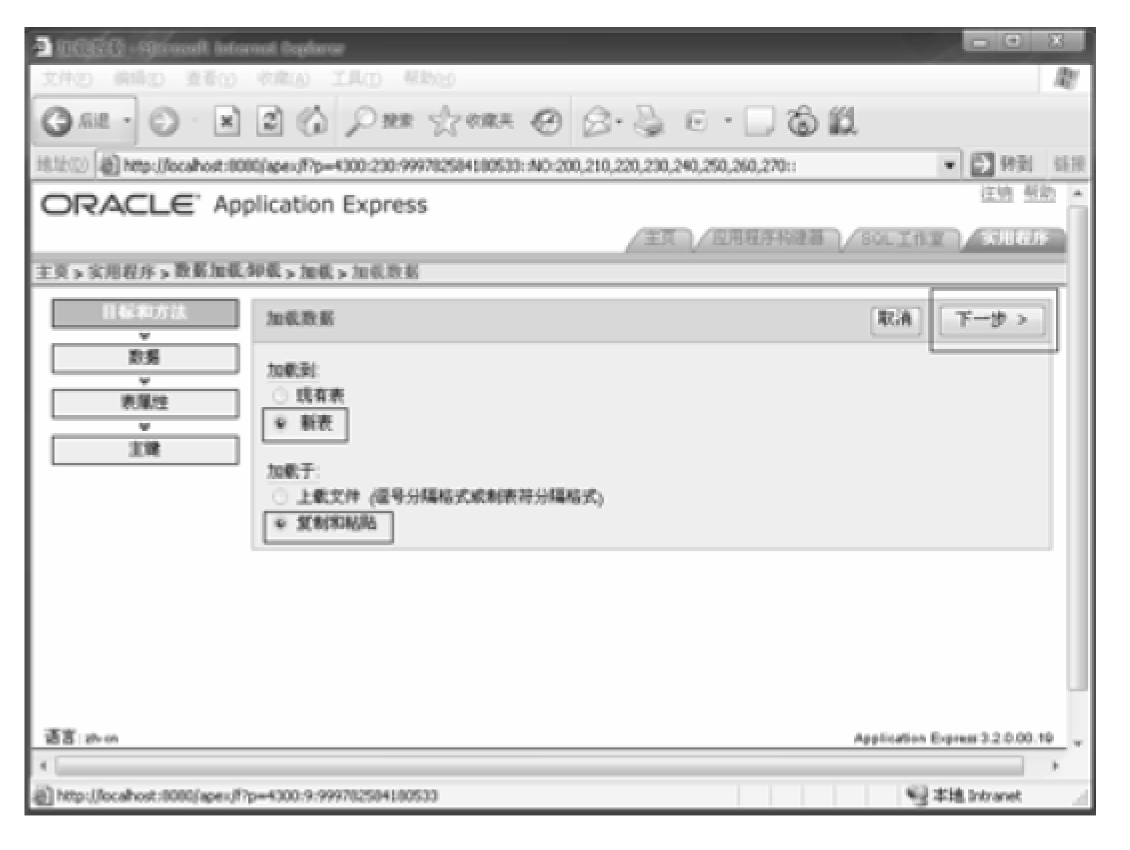


图　23.37

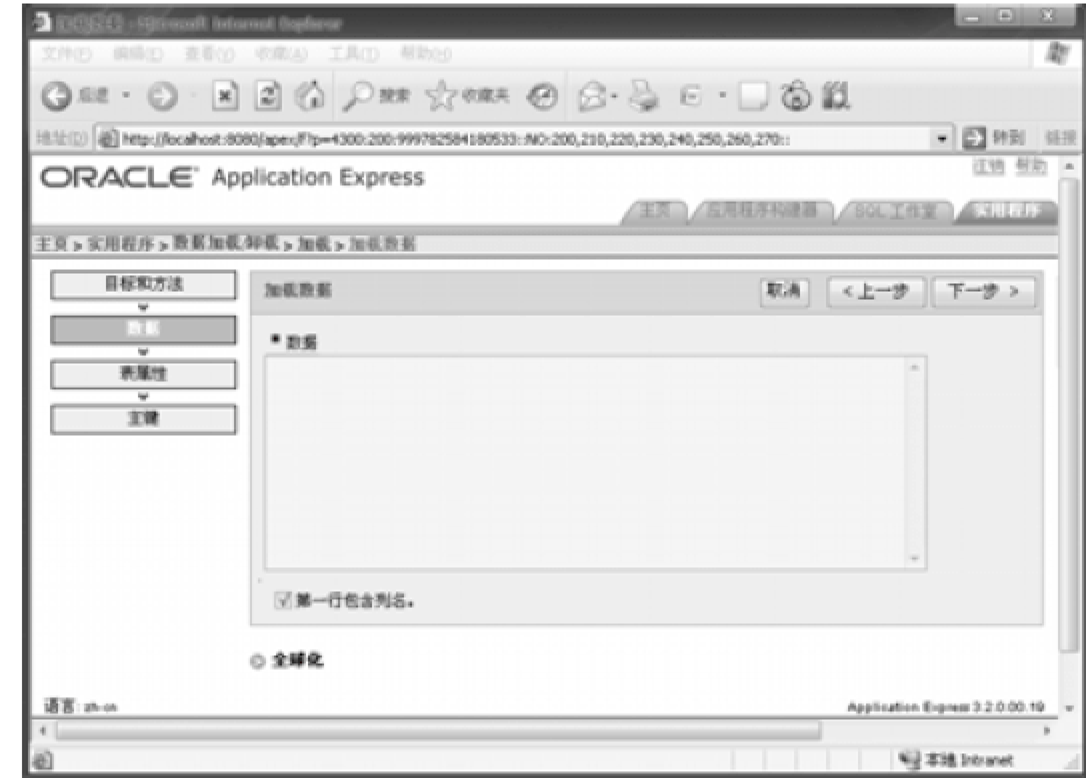


图　23.38

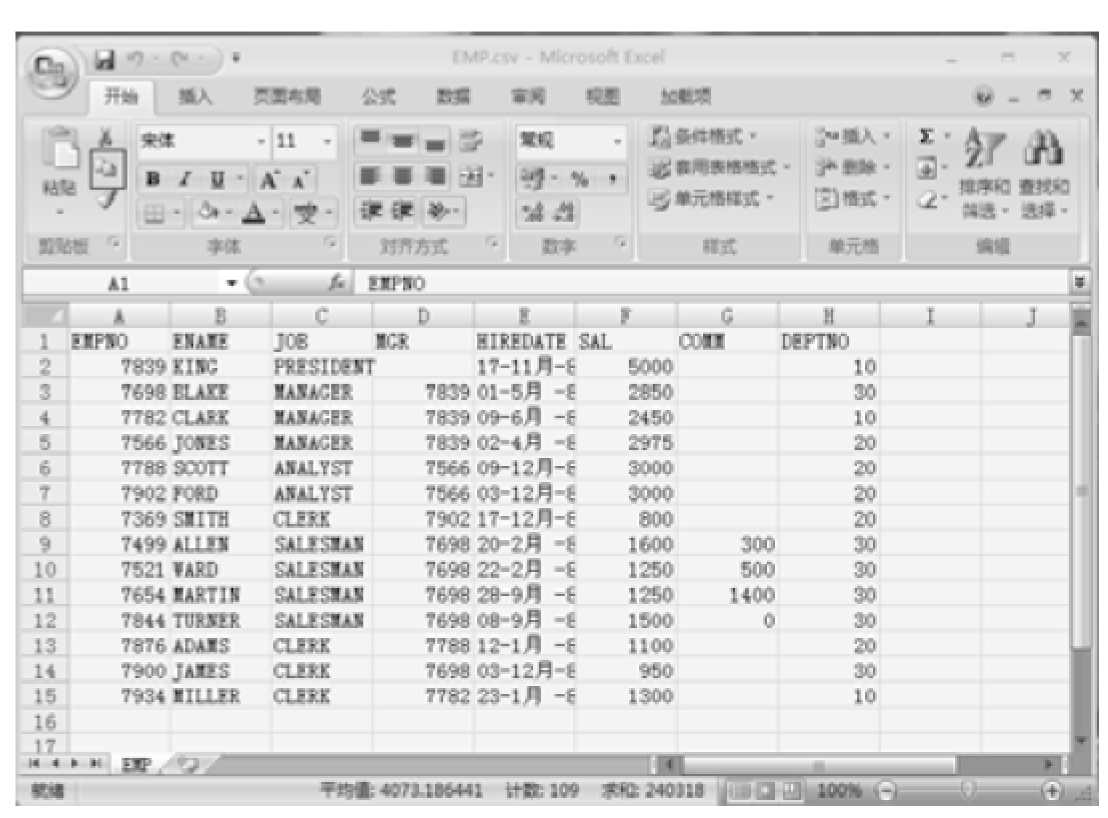
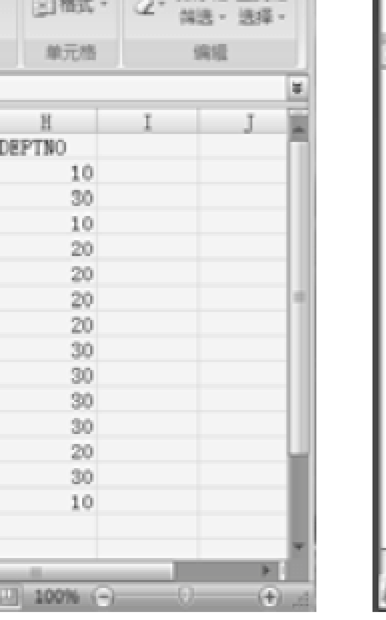



图　23.39



图　23.40

（6）在“表名”文本框中输入您认为合适的表名（这里为了方便输入 EMP2），其他选项接受默认设置，单击“下一步”按钮，如图 23.41 所示。

（7）在“主键来自”栏中选中“使用现有列”单选按钮，在“主键”下拉列表框中选择 EMPNO(NUMBER)选项，其他选项接受默认设置，单击“加载数据”按钮，如图 23.42 所示。之后将进入“电子表格资料档案库”页面。



图　23.41



图　23.42

（8）单击 EMP2 超链接，如图 23.43 所示。之后将进入“对象浏览器”页面。

（9）在“对象浏览器”页面中，您可以看到 COMM 的数据类型为变长字符型（最大长度为 30 个字符），如图 23.44 所示。这显然是有问题的，因此还需要修改。在一些情况下，默认值并不能满足实际应用的需要，是需要开发人员进行调整的。

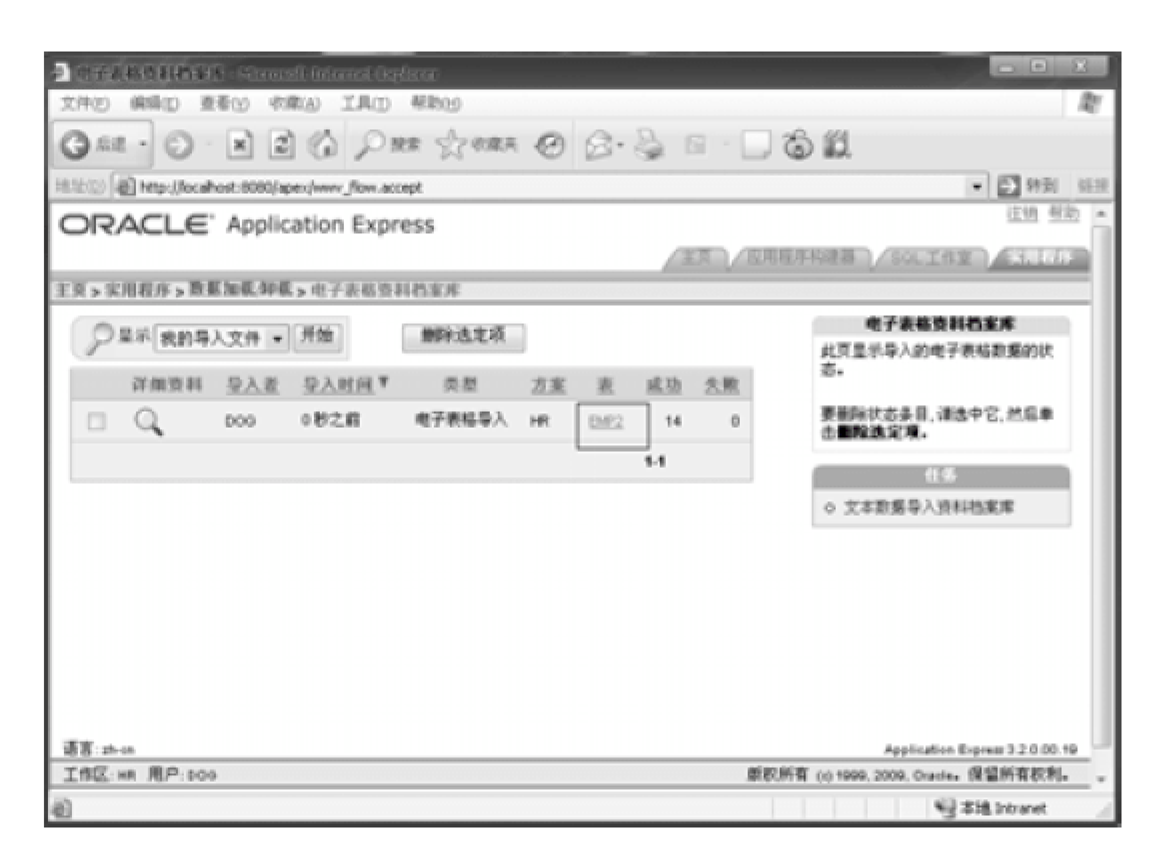


图　23.43

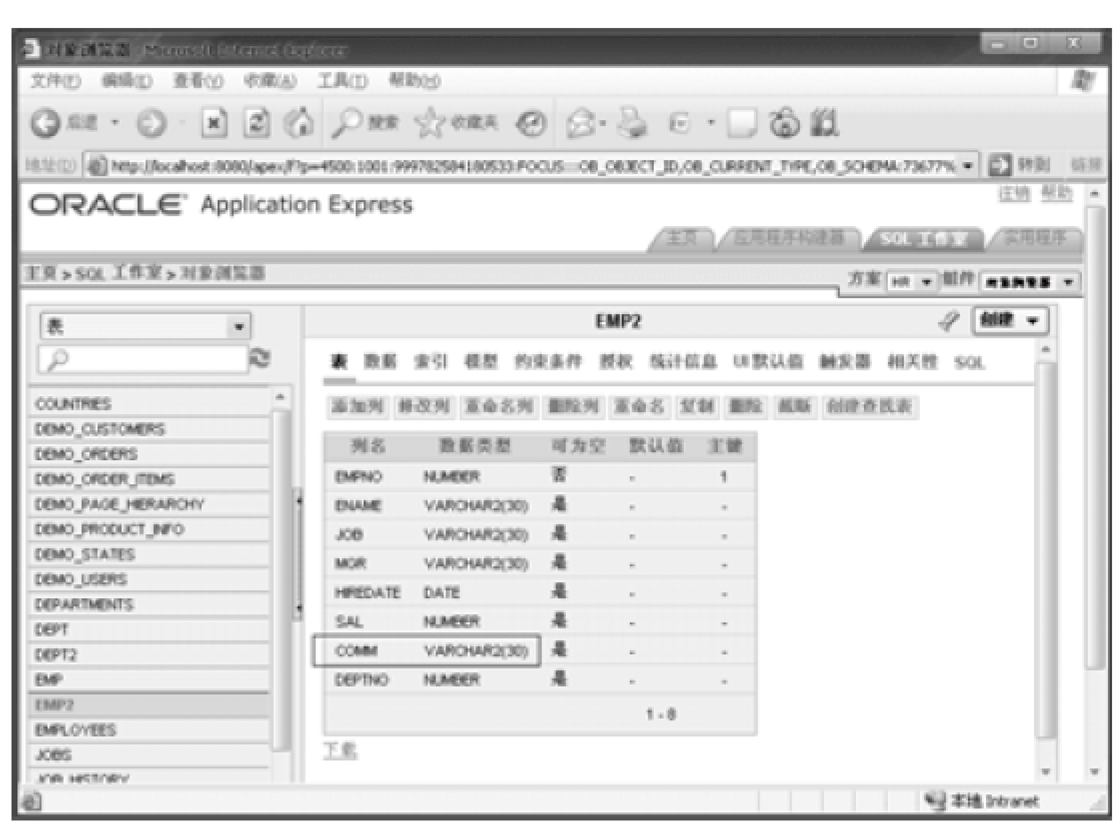


图　23.44

（10）此时，单击“数据”超链接就可以看到所上传的电子表格数据，如图 23.45 所示。

以上我们演示了使用复制和粘贴加载电子表格数据的操作，下面我们介绍另一种方法，就是使用上传文件加载电子表格数据，具体操作步骤如下：

（1）进入“加载”页面，单击“加载电子表格数据”图标，如图 23.46 所示。之后将进入“加载数据”页面。

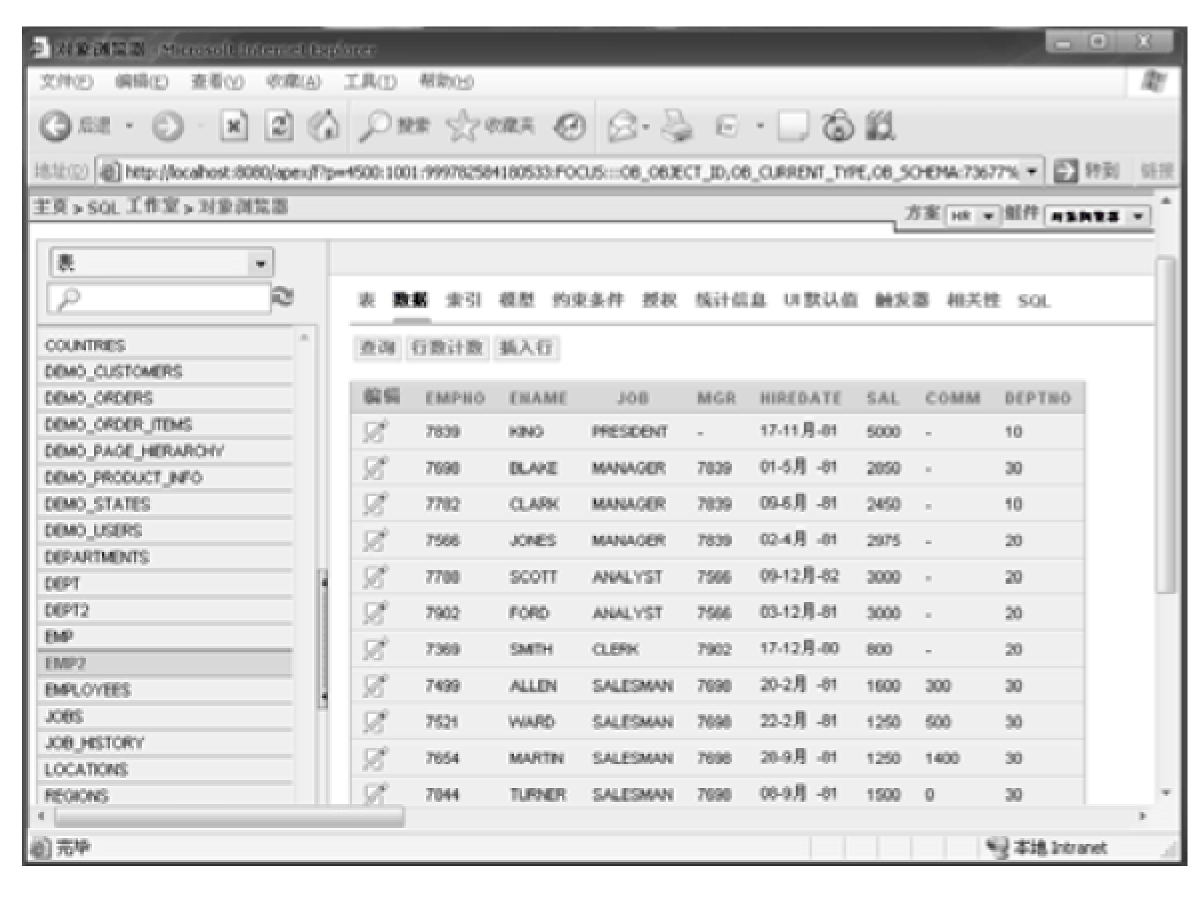


图　23.45

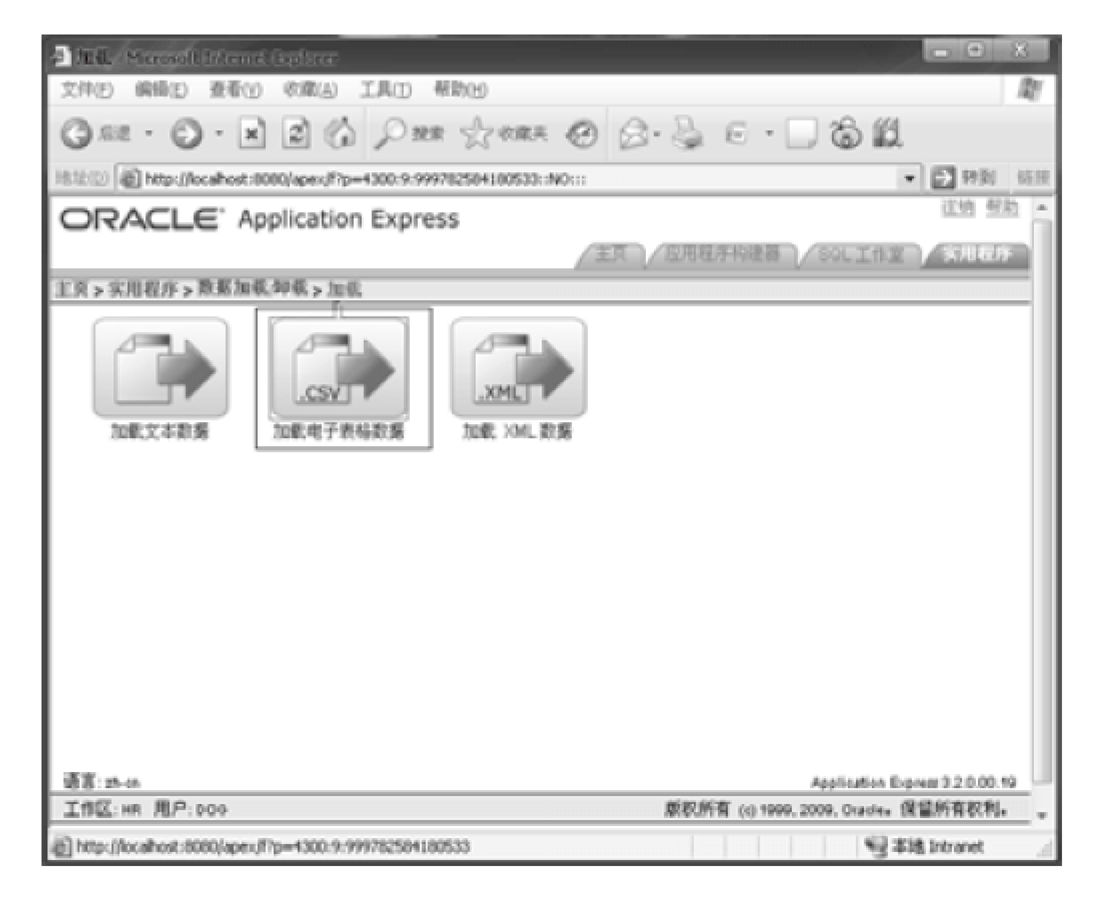


图　23.46

（2）在“加载到”栏中选中“新表”单选按钮，在“加载于”栏中选中“上载文件”单选按钮，单击“下一步”按钮，如图 23.47 所示。

（3）单击“文本文件”文本框右侧的“浏览”按钮，如图 23.48 所示。之后出现“选择文件”对话框。

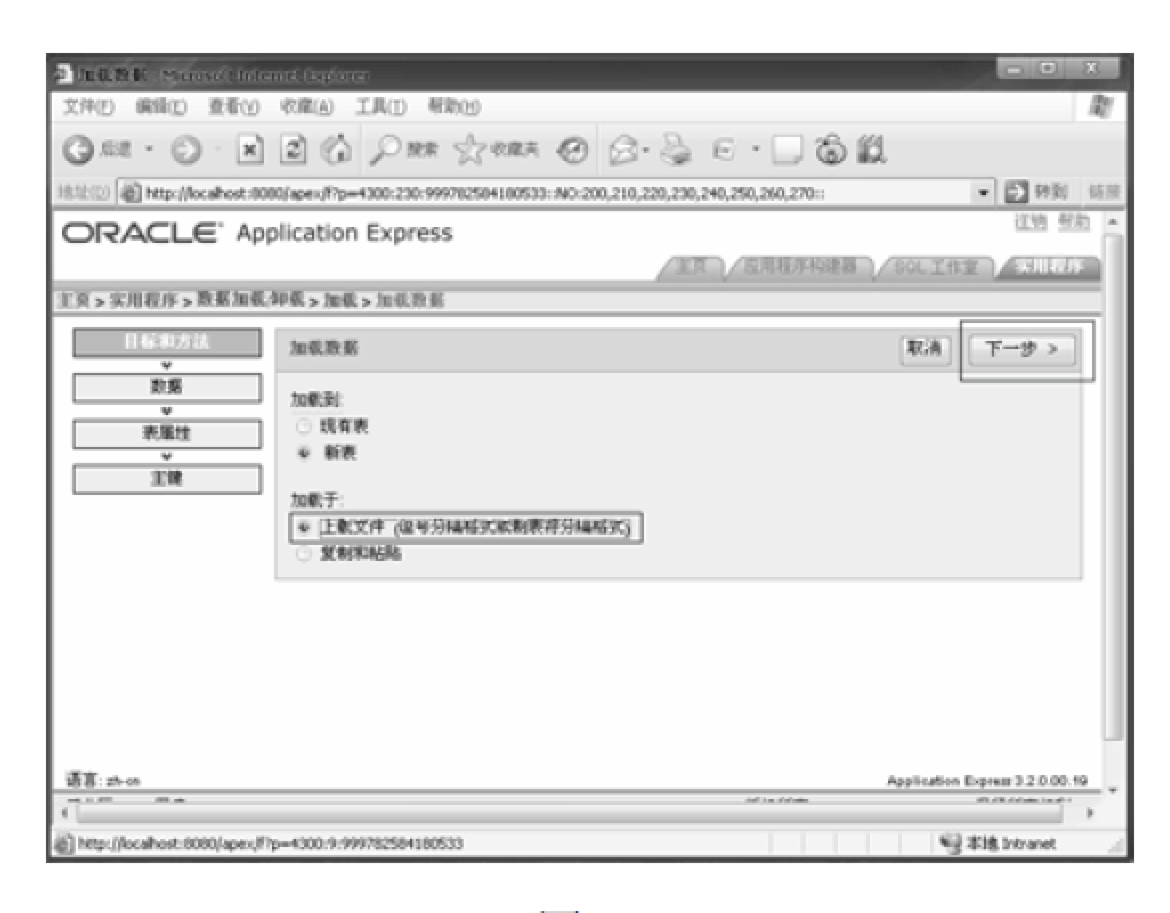


图　23.47

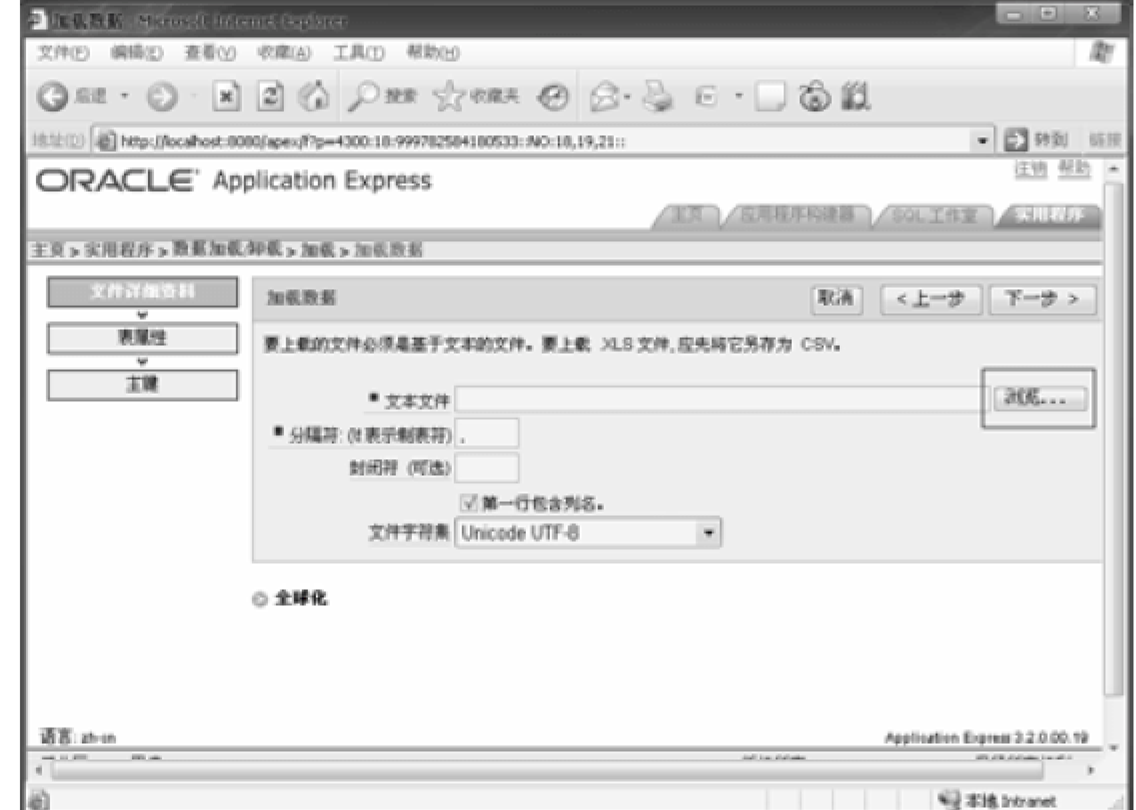


图　23.48

（4）在“查找范围”下拉列表框中选择 F 盘上的 Express_Data 目录（文件夹），在“文件名”下拉列表框中选择 EMP.csv，单击“打开”按钮，如图 23.49 所示。

（5）选中“第一行包含列名。”复选框，单击“下一步”按钮，其他接受默认设置，如图 23.50 所示。

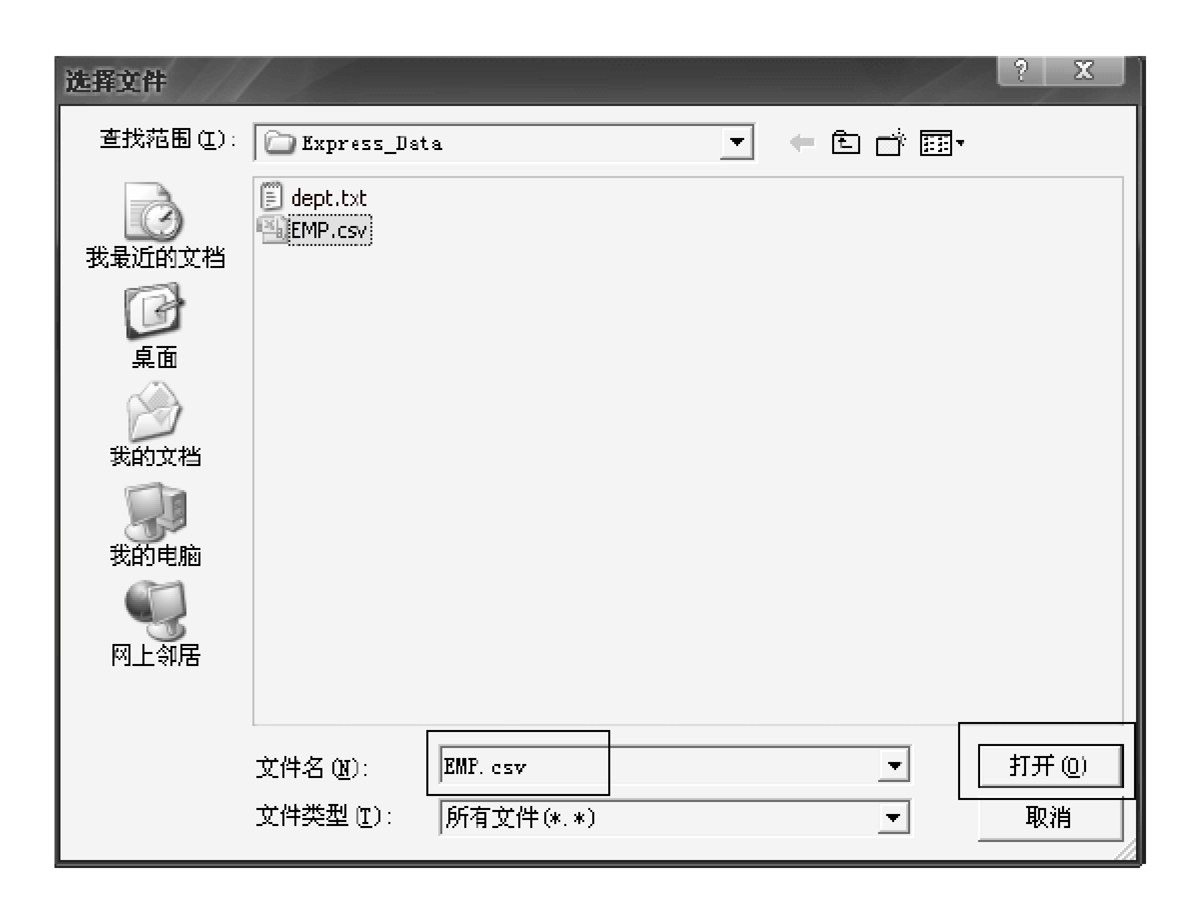


图　23.49

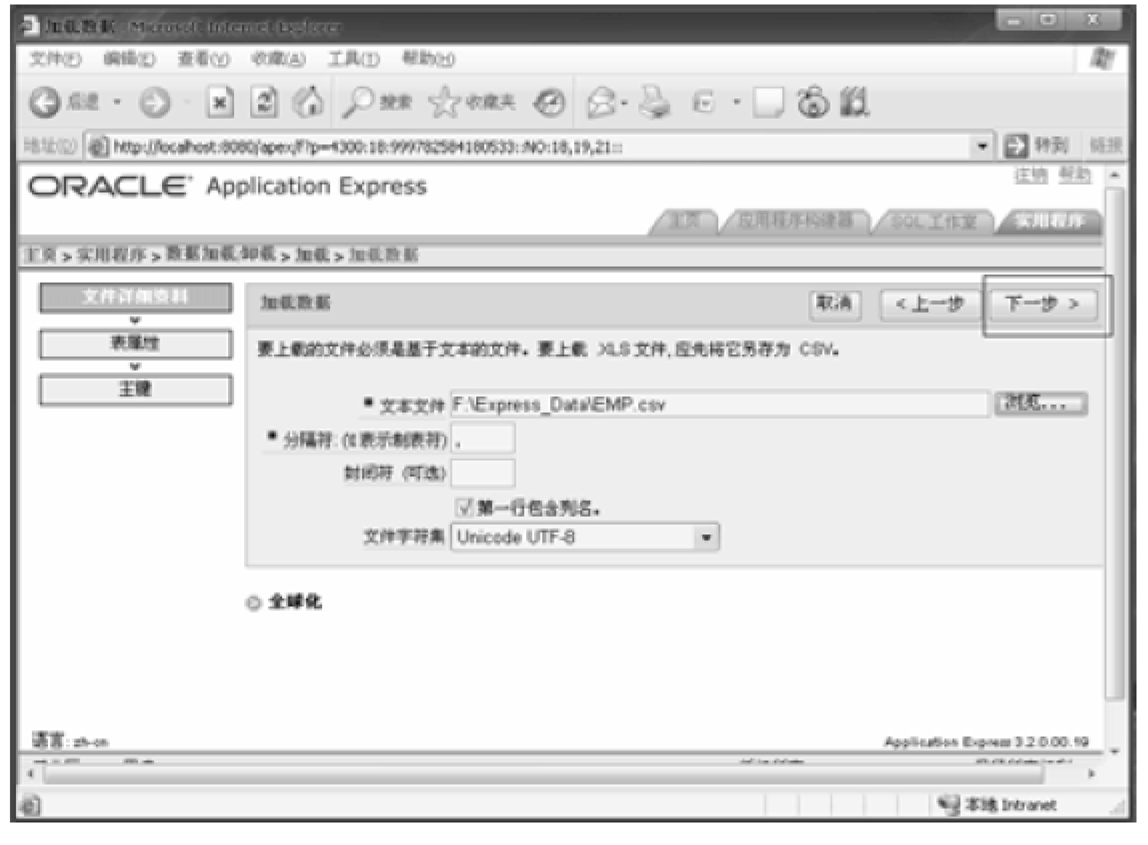


图　23.50

（6）此时，您会发现 HIREDATE 列的数据中包含了问号，这是字符集转换的问题，为此单击“上一步”按钮退回到上一页，如图 23.51 所示。

（7）在“文件字符集”下拉列表框中选择“中文 GBK”选项，单击“下一步”按钮，如图 23.52 所示。

现在您会发现 HIREDATE 列的数据已经没有问题了，如图 23.53 所示。后面的操作与使用复制和粘贴的方法加载电子表格数据的操作完全相同，在这里就不再重复了，有兴趣的读者可以自己试一下。

原来将商业智能（BI）软件所产生的数据上传给 Oracle 数据库也是那么容易。现在如果在面试时有人问您会不会将 BI 的数据导给 Oracle 数据库，您可以自信地回答“没问题。”



图 23.51

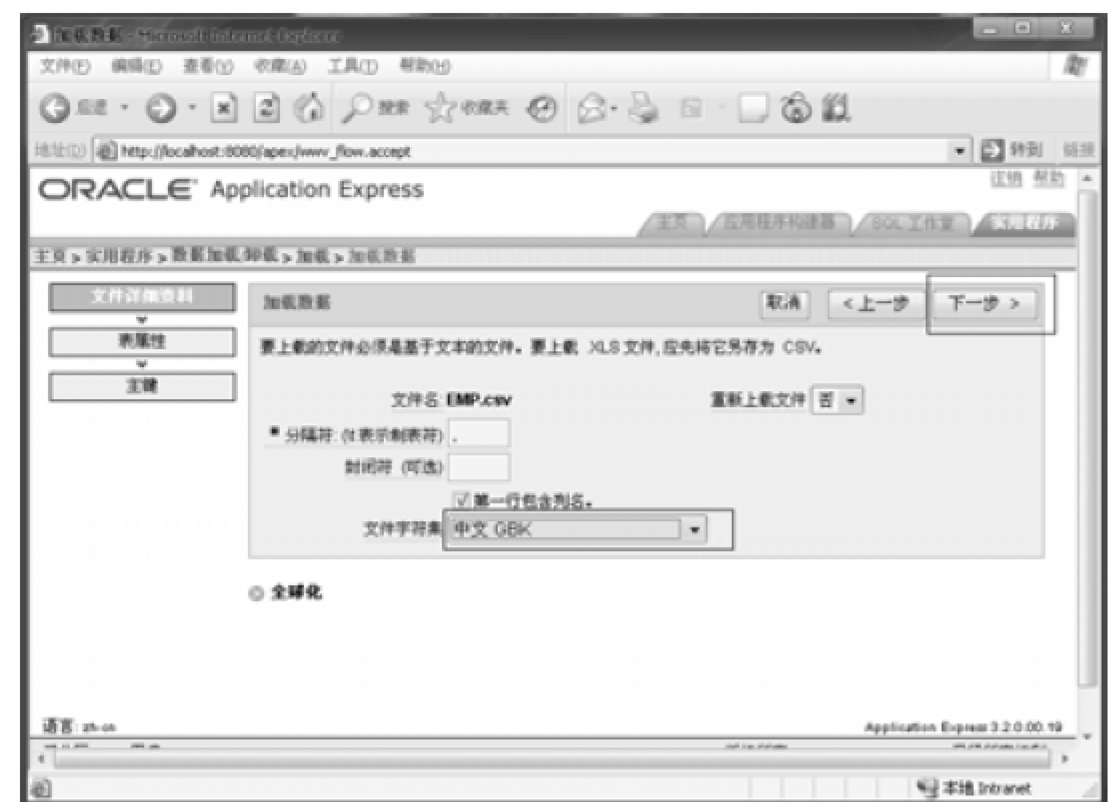


图 23.52

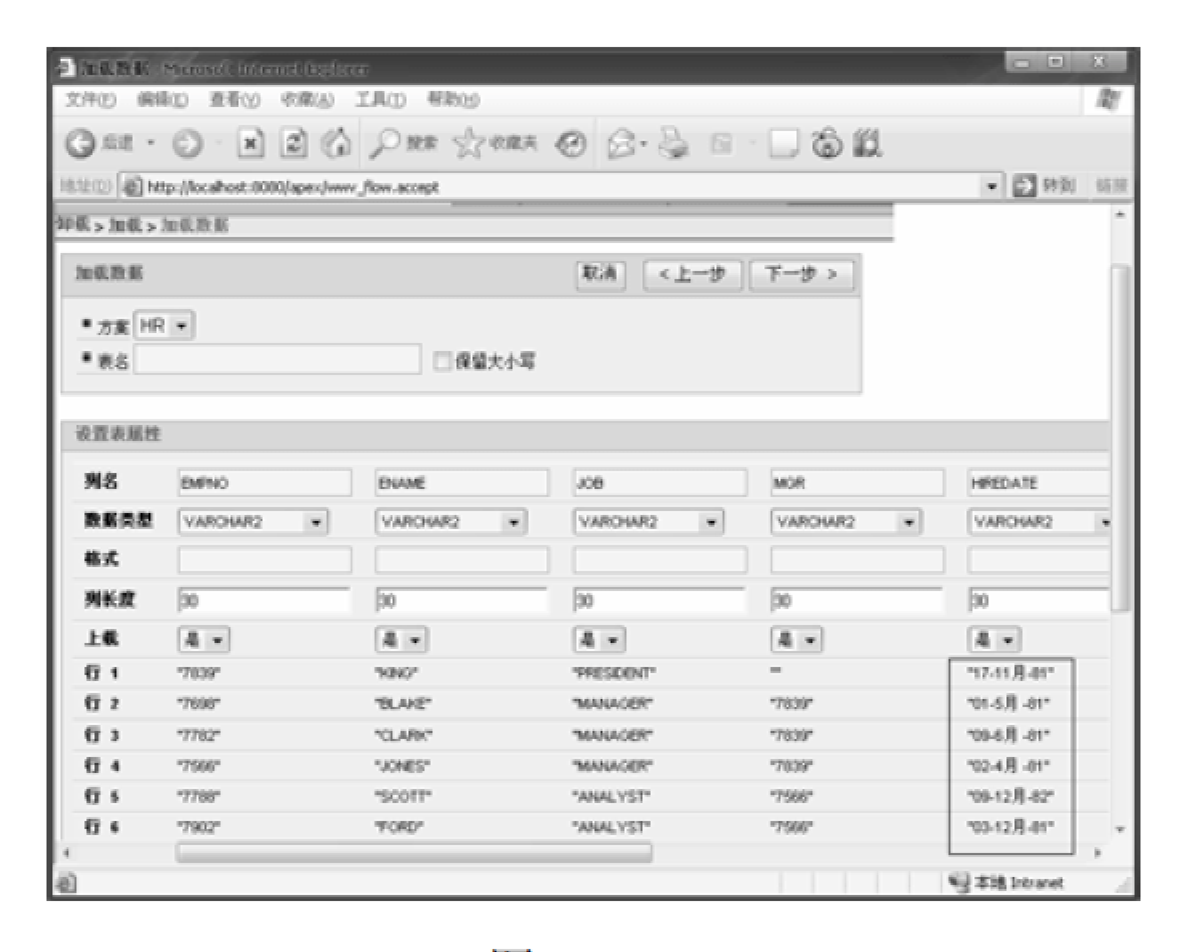


图 23.53

23.6 部署应用程序原理

当在开发环境中创建了应用程序之后，就需要部署该应用程序以便终端用户可以开始使用它。图 23.54 为在 Oracle Application Express 中部署应用程序原理的示意图。

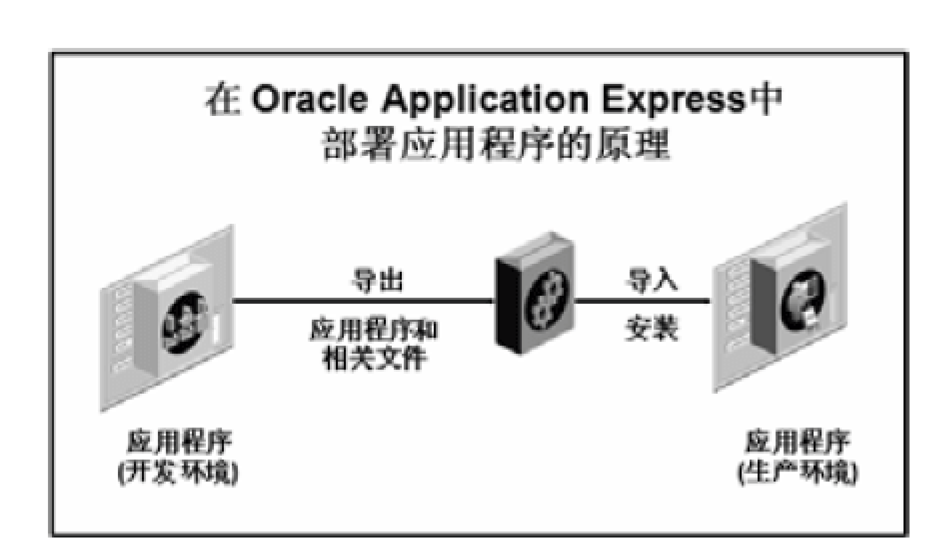


图 23.54

生产环境可以是与开发环境不同的计算机、不同的操作系统，其硬件配置也可能有很多的差别。要将所开发的应用程序从开发环境转移到生产环境，就必须导出应用程序，之

后再将它导入到生产实例中。要注意的是要转移的不仅仅是应用程序，还可能包括相关的文件和对象等。

要进行导入/导出操作，首先进入“应用程序”页面，之后单击“导出/导入”图标，如图 23.55 所示。

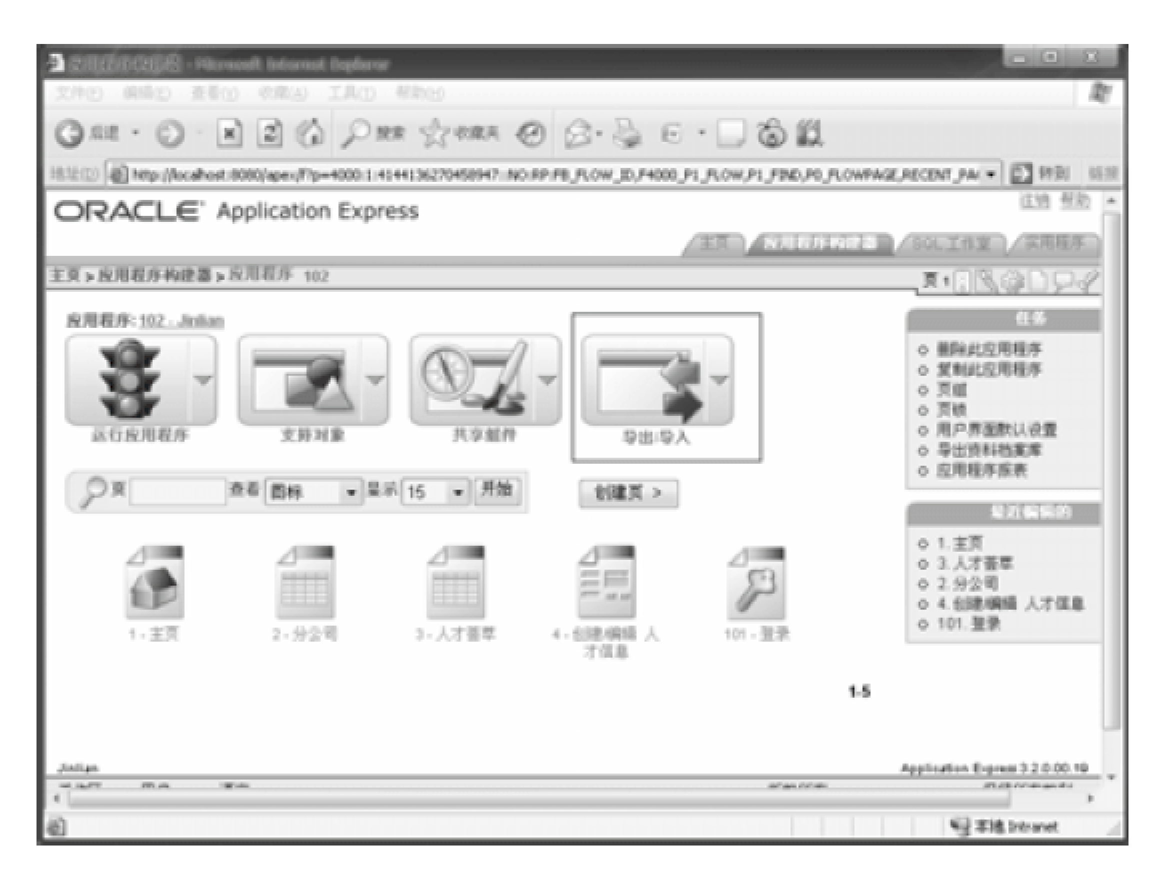


图　23.55

23.7　导出应用程序

接下来我们将详细介绍如何导出在开发环境中创建的应用程序 Jinlian，具体操作步骤如下：

（1）在“导出/导入”页面中选中“导出”单选按钮，单击“下一步”按钮，如图 23.56 所示。之后将进入“导出”页面。

（2）单击某个您感兴趣的超链接，如调试等，如图 23.57 所示。之后将出现“调试”窗口。

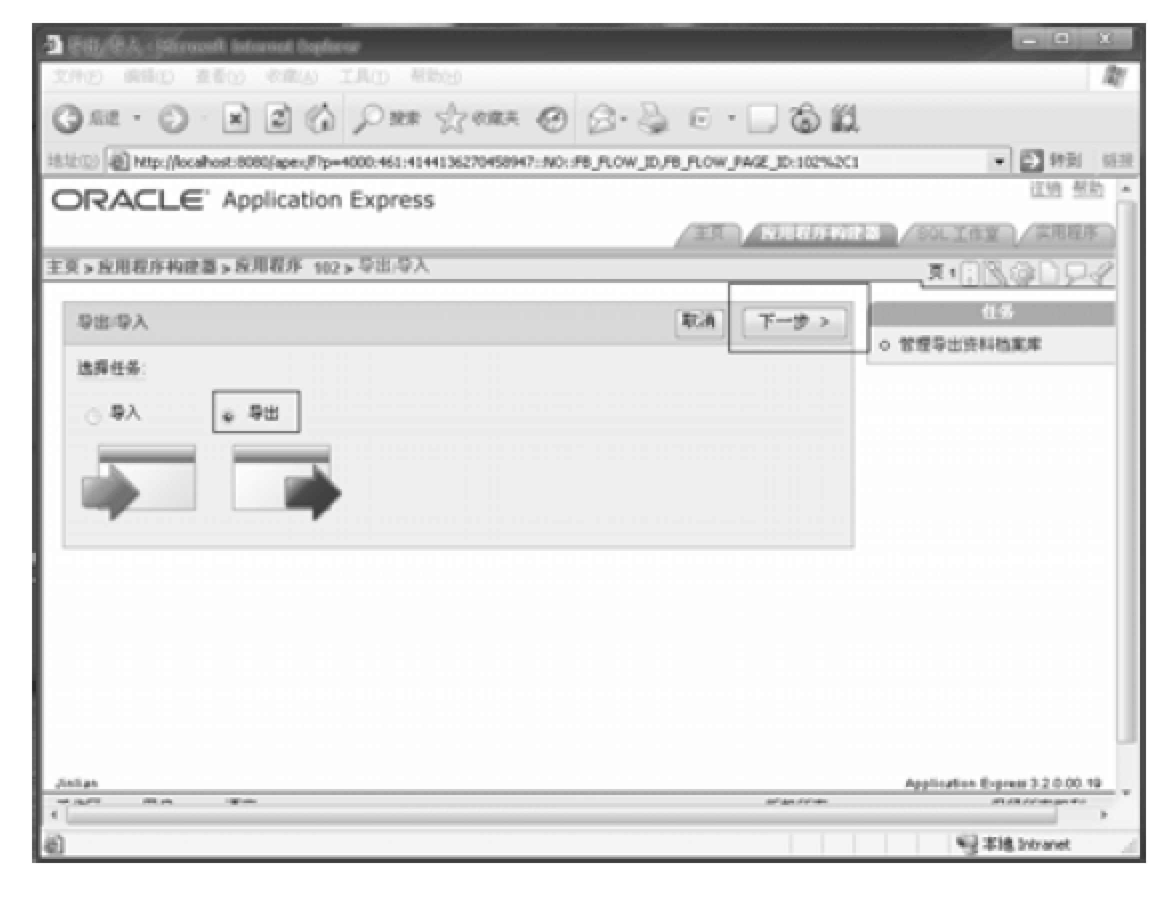


图　23.56



图　23.57

（3）如果在阅读了说明之后还不能理解，可以单击“帮助”超链接，如图 23.58 所示。

之后将打开应用程序构建器用户指南，您可以慢慢地阅读了，不过前提是您能看懂英文。

（4）关闭如图 23.59 所示的“应用程序构建器用户指南”窗口和“调试”窗口，退回到“导出”页面。

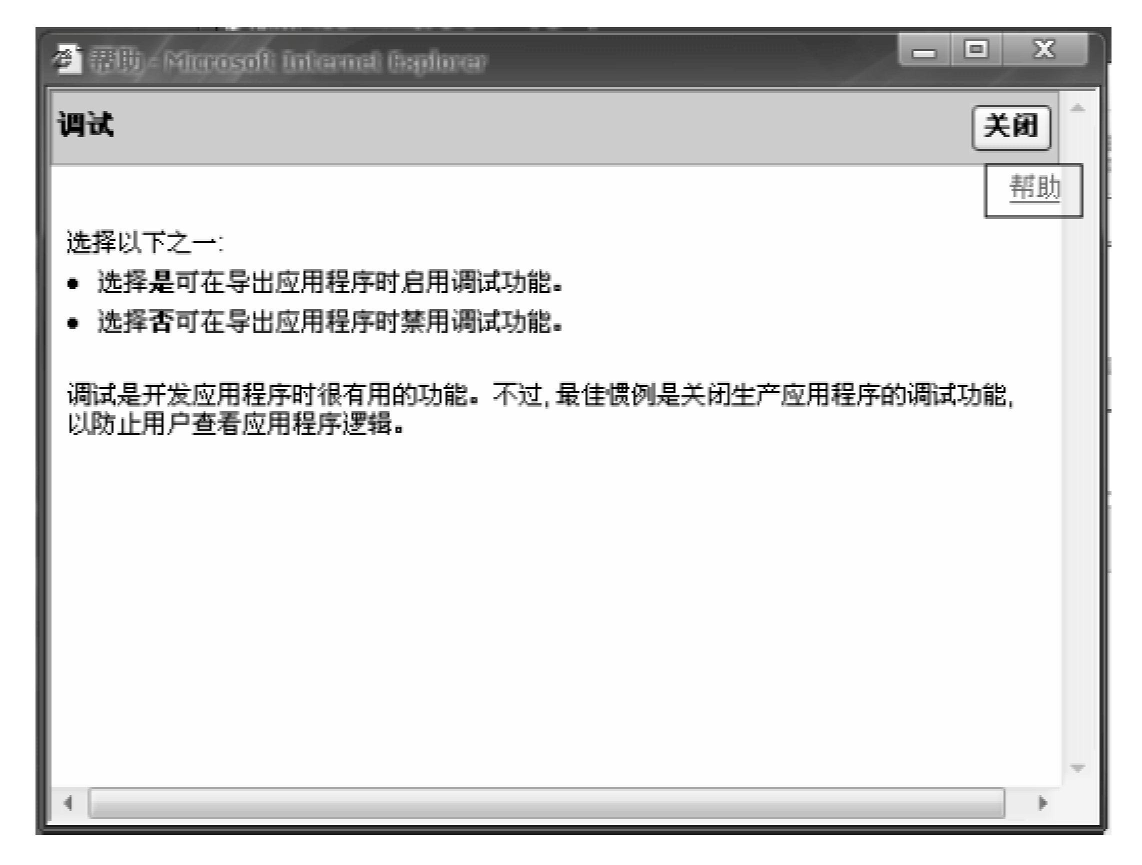


图 23.58



图 23.59

（5）选择文件格式，在“调试”下拉列表框中选择“否”，其他选择“是”，其余选项接受默认设置，单击“导出应用程序”按钮，如图 23.60 所示。之后将出现“文件下载”对话框。

（6）单击“保存”按钮，如图 23.61 所示。之后将出现“另存为”对话框。

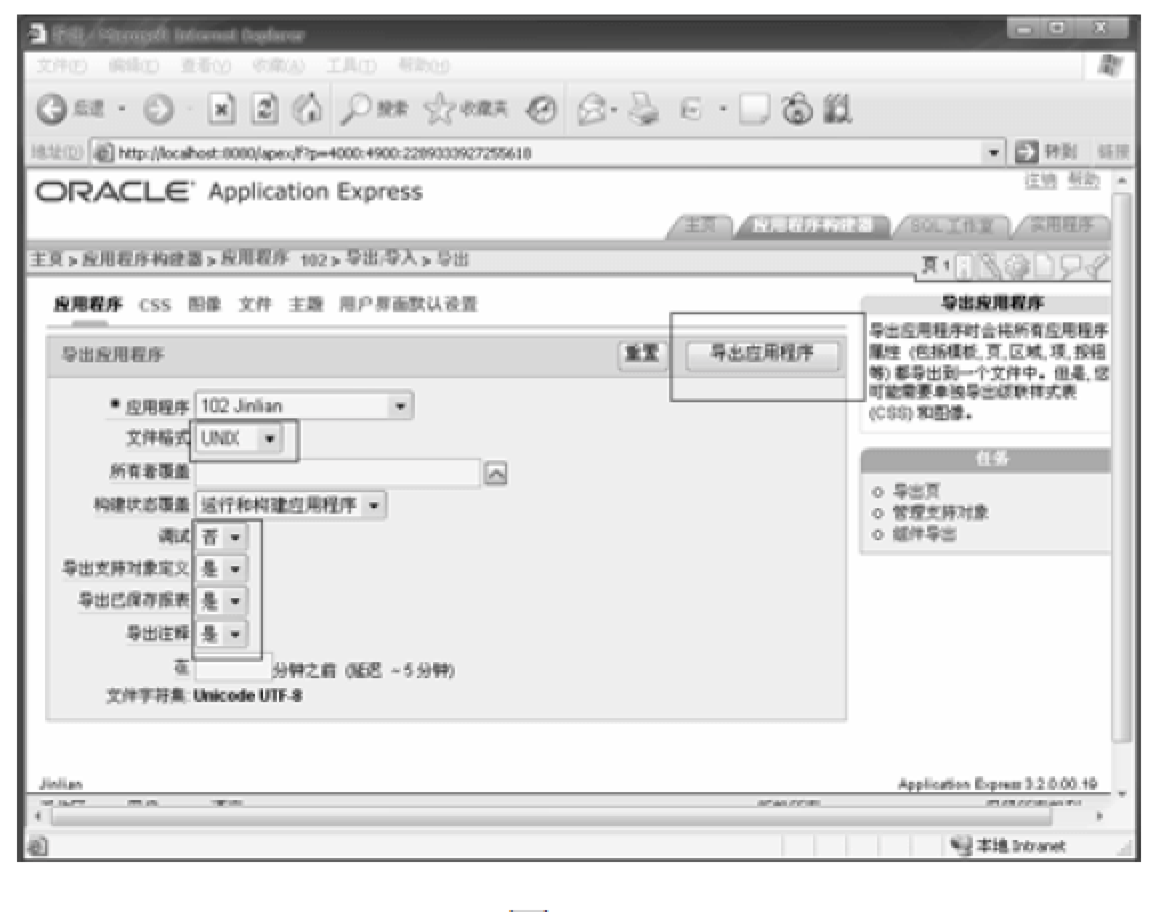


图 23.60



图 23.61

（7）在“保存在”下拉列表框中选择 F 盘上的 Express_Data 目录（文件夹），其他选项接受默认设置，单击“保存”按钮，如图 23.62 所示。

（8）进入 Express_Data 目录（文件夹），您会发现多了一个刚生成的 f102.sql 文件，如图 23.63 所示。

从图 23.63 的显示结果中就可以确认您已经成功地导出了在开发环境中创建的应用程序 Jinlian。



图　23.62



图　23.63

23.8　下载客户追踪包和创建安装所用的工作区

为了演示导入应用程序的操作过程，我们从 Oracle Application Express 的官方网站上下载一个名为 customer_tracker_1.1.zip 的软件包（也可以下载其他软件包）。首先将您的计算机连接到互联网上，启动网络浏览器并输入如下的网址：

http://www.oracle.com/technology/products/database/application_express/packaged_apps/packaged_apps.html

在这个网站上有许多常用的软件包，可以免费下载，然后安装到您的系统上并加以修改。这样可以减少您的开发时间，同时也可以迅速地扩展您的 Express 知识和技能。

下载完成后要将该压缩包解压缩，解压之后会产生两个文件，一个是 SQL 脚本文件，另一个是正文的 readme 文件。您应该先阅读一下 readme 文件，然后将这两个文件复制到安装目录。为了方便，此时将整个 customer_tracker_1.1 目录复制到 F:\Express_Data 目录下，如图 23.64 所示。



图　23.64

不要在现有的 HR 工作区上安装这个应用程序软件包，因为可能会有一些对象发生冲突，所以最安全、也是最简单的方法是为安装这一软件包创建一个新的工作区。以下就是创建这个工作区的具体操作步骤：

（1）启动网络浏览器，将 http://localhost:8080/apex/apex_admin 输入到地址栏以登录 Oracle Application Express 管理服务。

（2）在登录界面的“用户名”文本框中输入“admin”，在“口令”文本框中输入“xm_Q1ng”，单击“登录”按钮，如图 23.65 所示。之后将出现“主页”页面。

（3）单击“管理工作区”图标，如图 23.66 所示。之后将进入“管理工作区”页面。

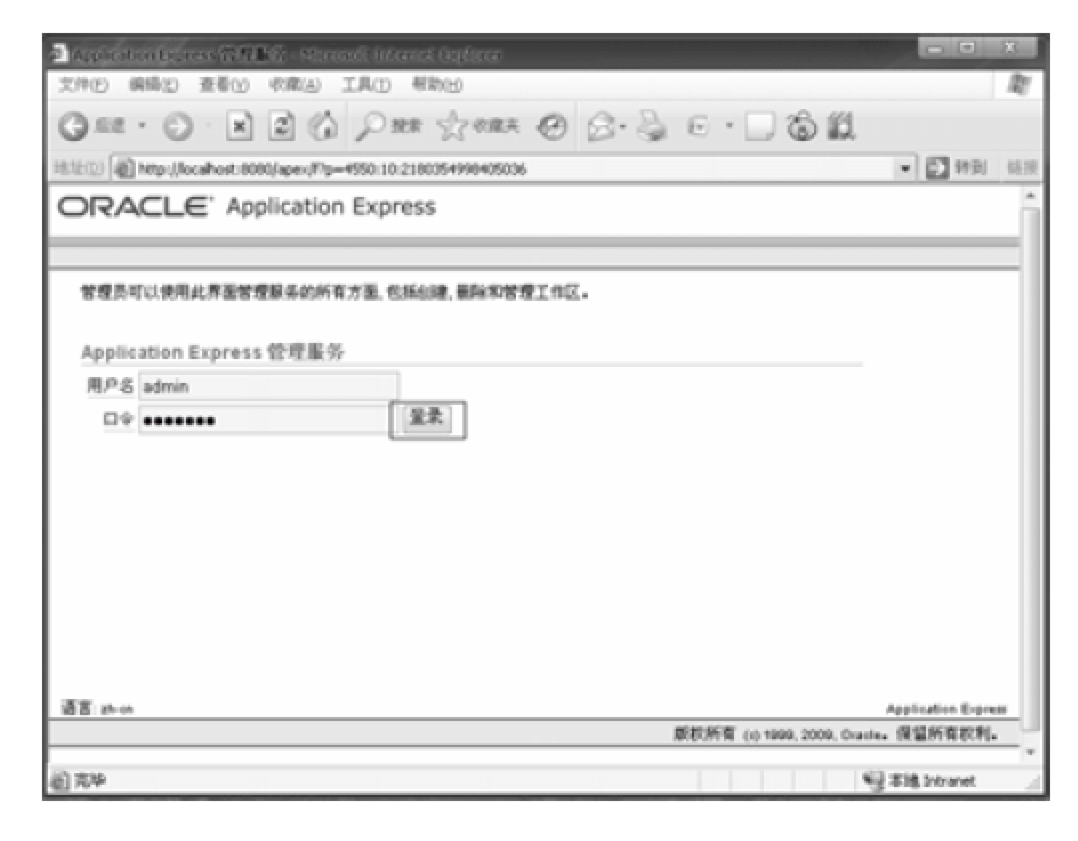


图　23.65



图　23.66

（4）在“管理工作区”区域单击“创建工作区”超链接，如图 23.67 所示。之后将进入“创建工作区”页面。

（5）为了简单起见，在“工作区名称”文本框中输入“cust”，在“工作区说明”列表框中输入您认为合适的解释信息，单击“下一步”按钮，如图 23.68 所示。



图　23.67

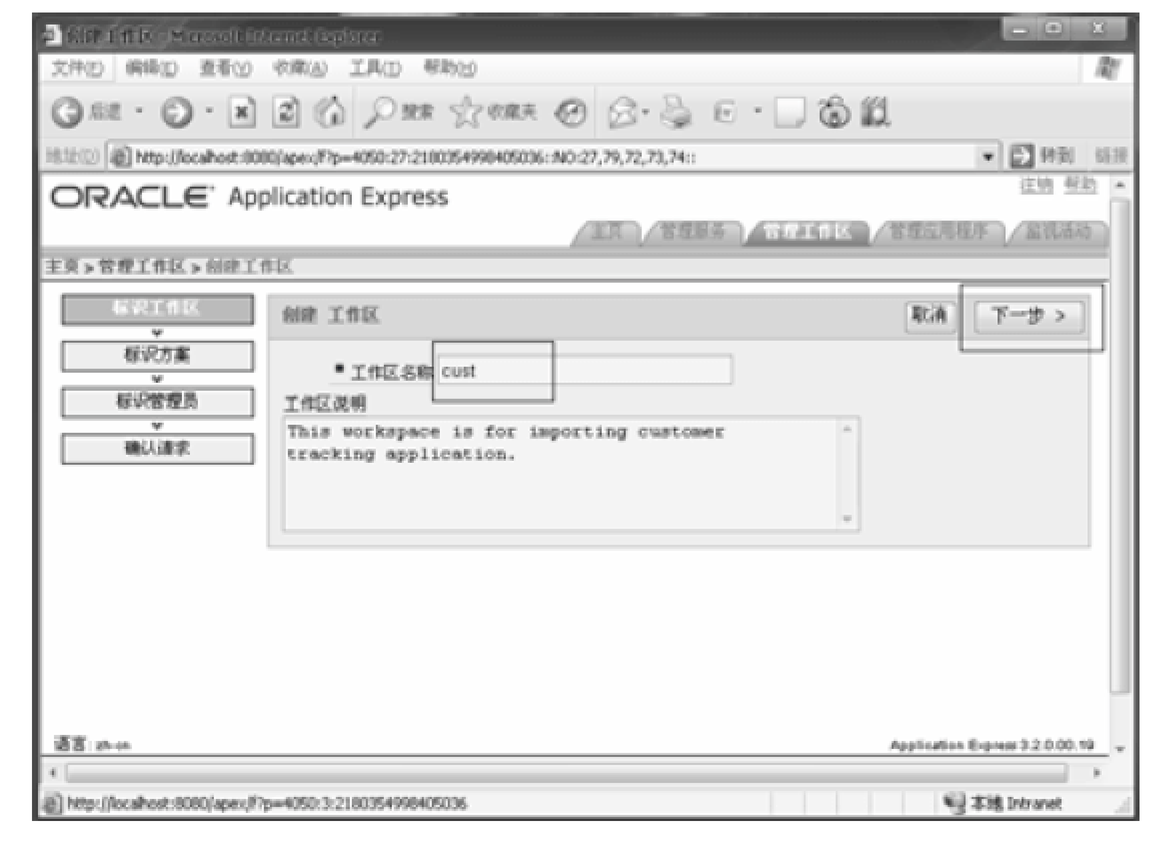


图　23.68

（6）在“是否重用现有方案？”下拉列表框中选择“否”，在“方案名”文本框中输入“cust”、在“方案口令”文本框中输入“xm_Q1ng”，在“空间限额”下拉列表框中选择 50MB，单击“下一步”按钮，如图 23.69 所示。

（7）在“管理员用户名”文本框中输入“ADMIN”，在“管理员口令”文本框中输入“xm_Q1ng”，在“名”文本框中输入“金莲”，在“姓”文本框中输入“潘”，在“电子邮件”文本框中输入“pjinlian@yahoo.com.cn”，单击“下一步”按钮，如图 23.70 所示。

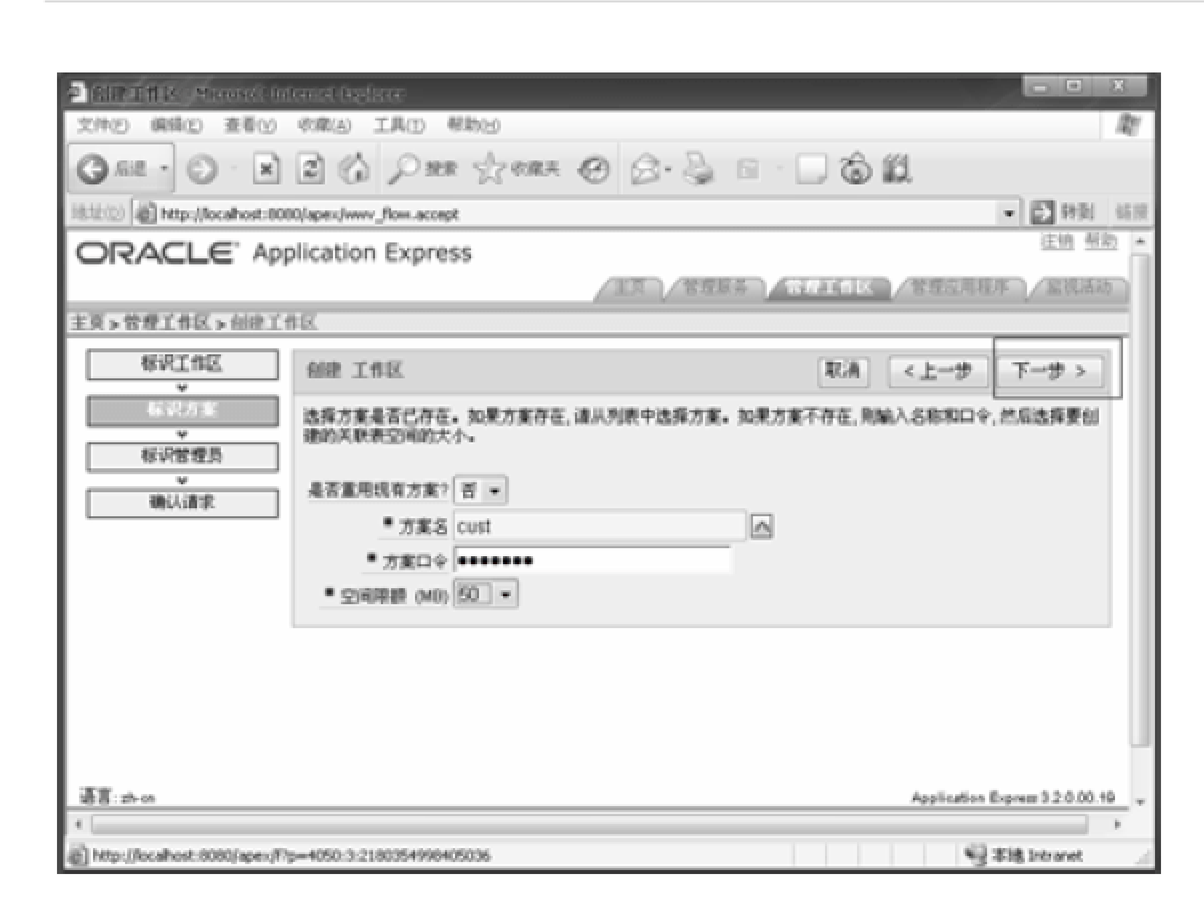


图　23.69

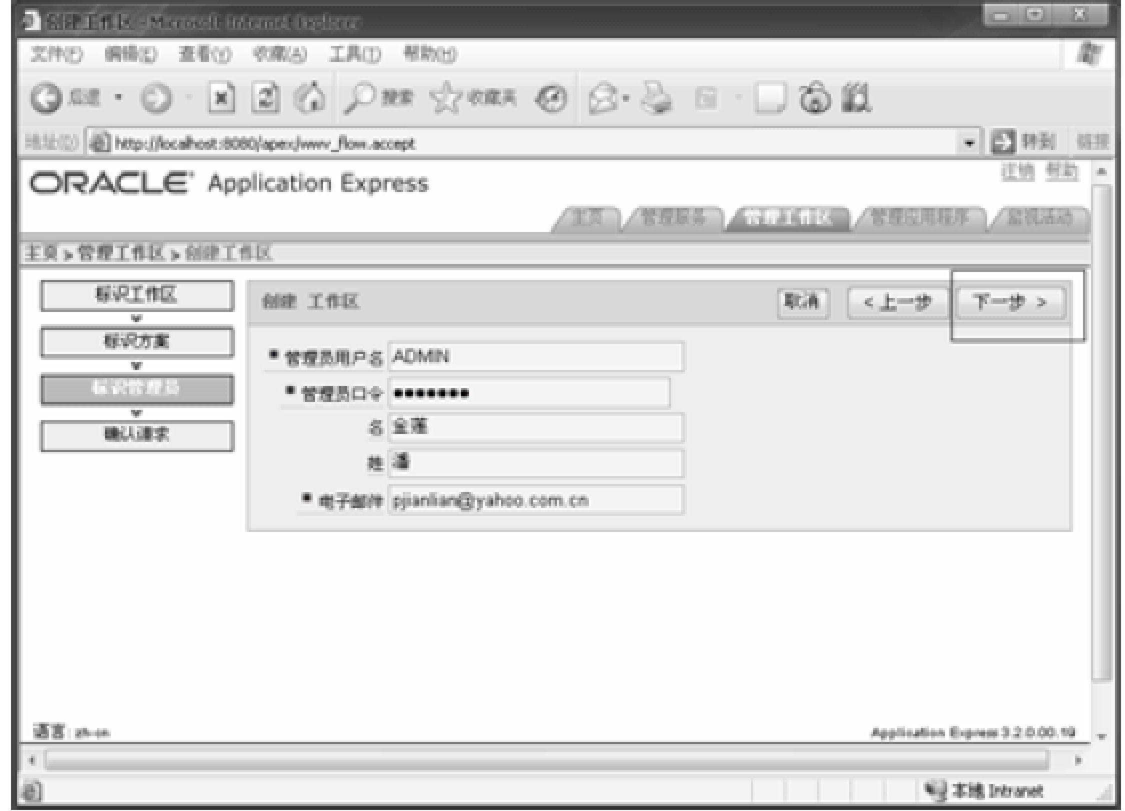


图　23.70

（8）检查所显示的信息，如果有问题要重新修改；如果没问题，单击“创建”按钮，如图 23.71 所示。

（9）单击“完成”按钮，如图 23.72 所示。之后将回到“管理工作区”页面。



图　23.71

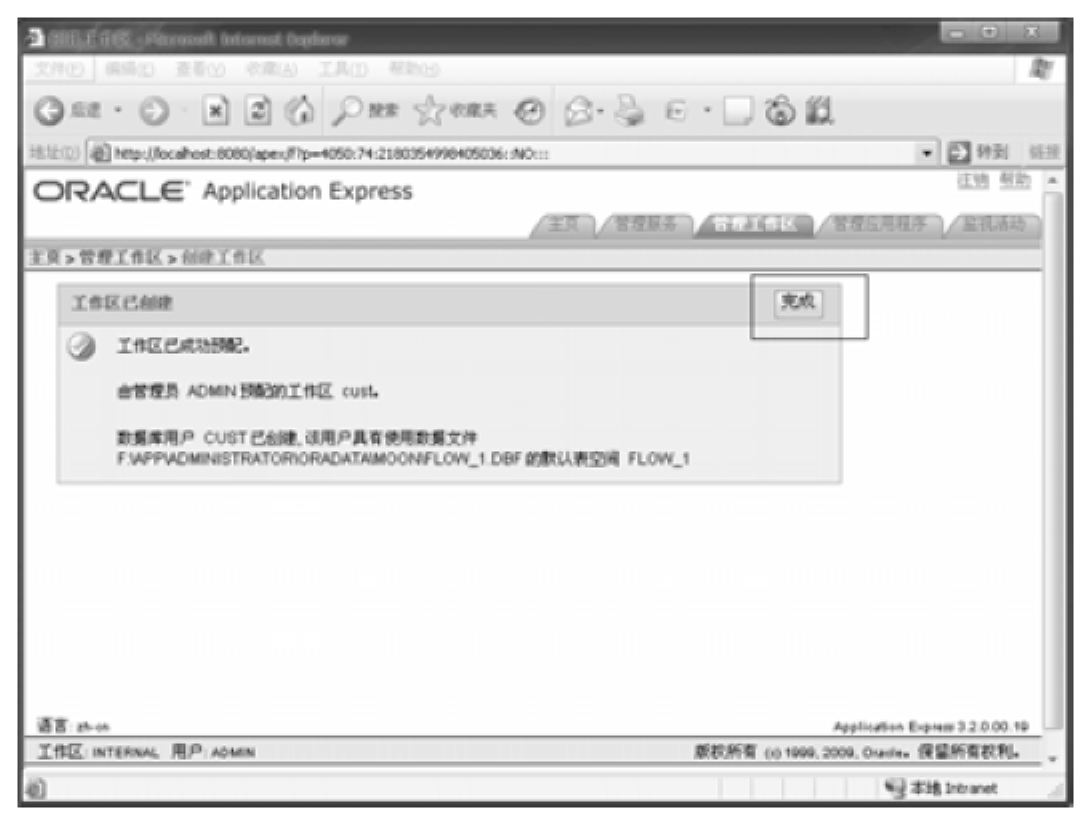


图　23.72

出现如图 23.73 所示的“管理工作区”页面后就表示您已经成功地创建了 cust 工作区，接下来就可以在这个工作区上安装所下载的应用程序软件包 customer_tracker_installer_ 1.1.sql 了。



图　23.73

23.9　在cust工作区上安装客户追踪软件包

在新创建的工作区 cust 上安装应用程序软件包 customer_tracker_installer_1.1.sql 的具体操作步骤如下：

（1）启动网络浏览器并在地址栏输入“http://localhost:8080/apex”，单击“转到”按钮。之后将进入 Oracle Application Express 的登录页面。

（2）在“工作区”文本框中输入“cust”，在“用户名”文本框中输入“ADMIN”，在“口令”文本框中输入“xm_Q1ng”，单击“登录”按钮，如图 23.74 所示。

（3）单击“应用程序构建器”图标，如图 23.75 所示，进入“应用程序构建器”页面。

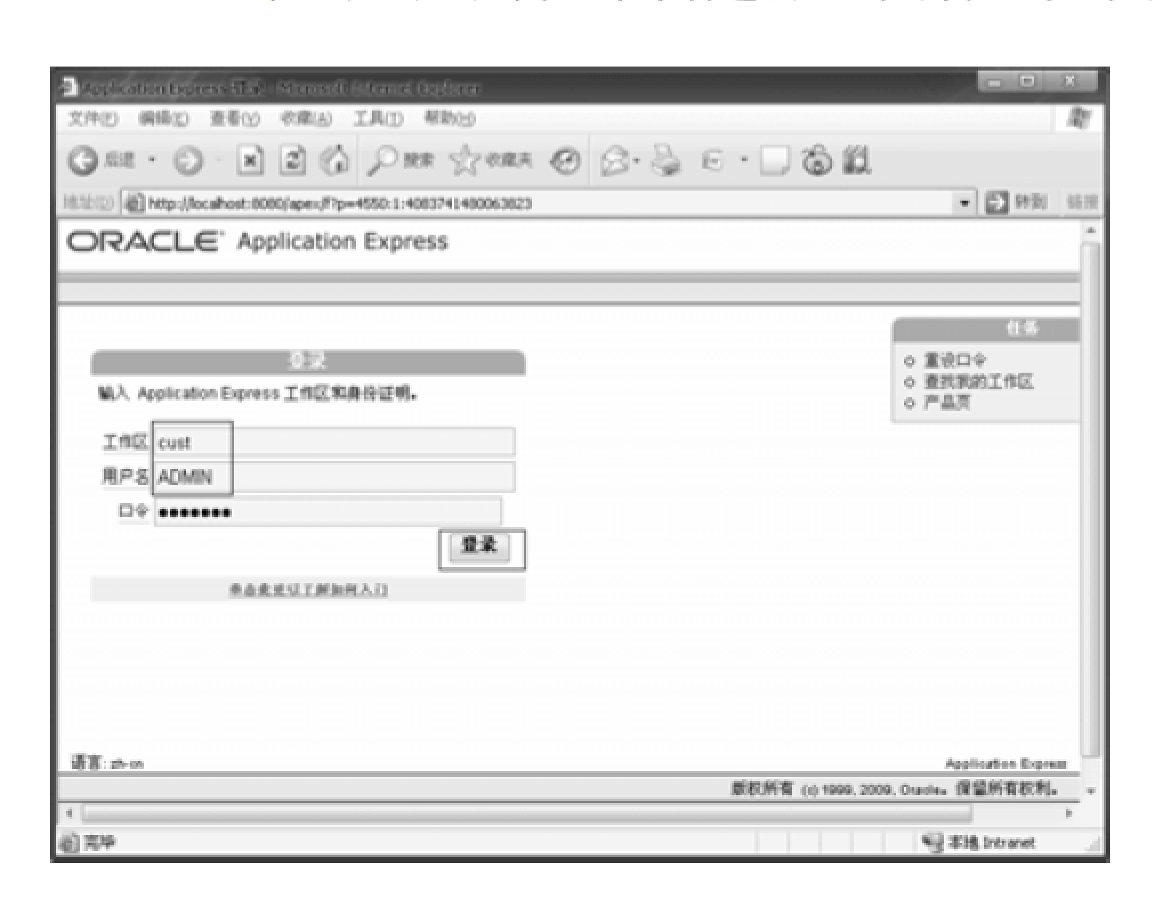


图　23.74

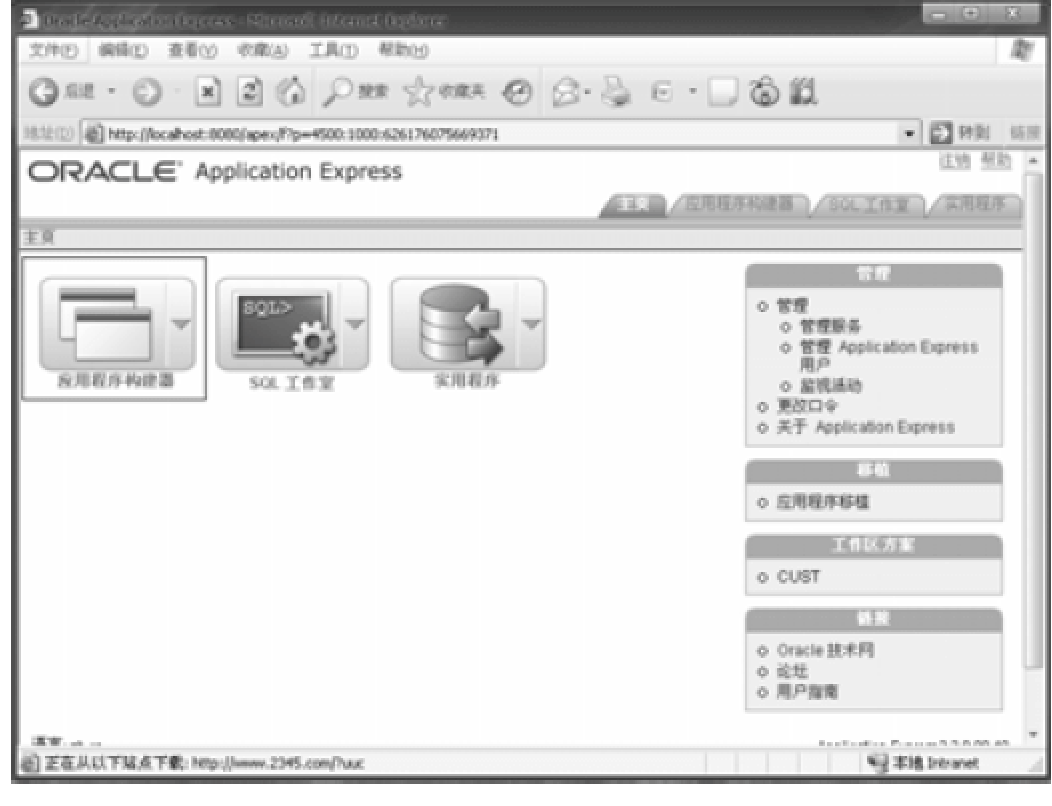


图　23.75

（4）单击 Sample Application - 107 图标（也可以单击“导入”按钮），如图 23.76 所示，进入“应用程序 107”页面。

（5）单击“导出/导入”图标，如图 23.77 所示，将进入“导出/导入”页面。

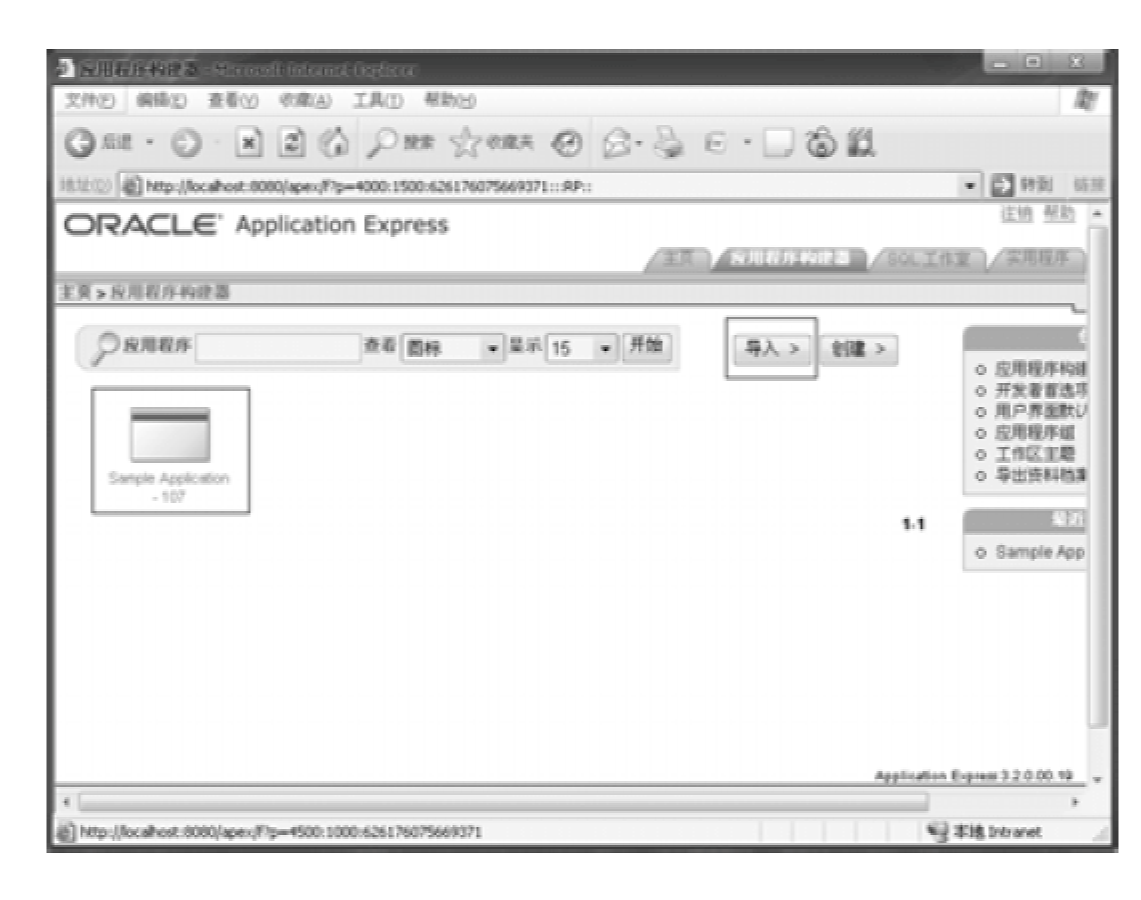



图　23.76



图　23.77

（6）选中“导入”单选按钮，单击“下一步”按钮，如图 23.78 所示。之后将进入“导

入”页面。

（7）单击“导入文件”文本框右侧的“浏览”按钮，如图 23.79 所示。之后出现“选择文件”对话框。

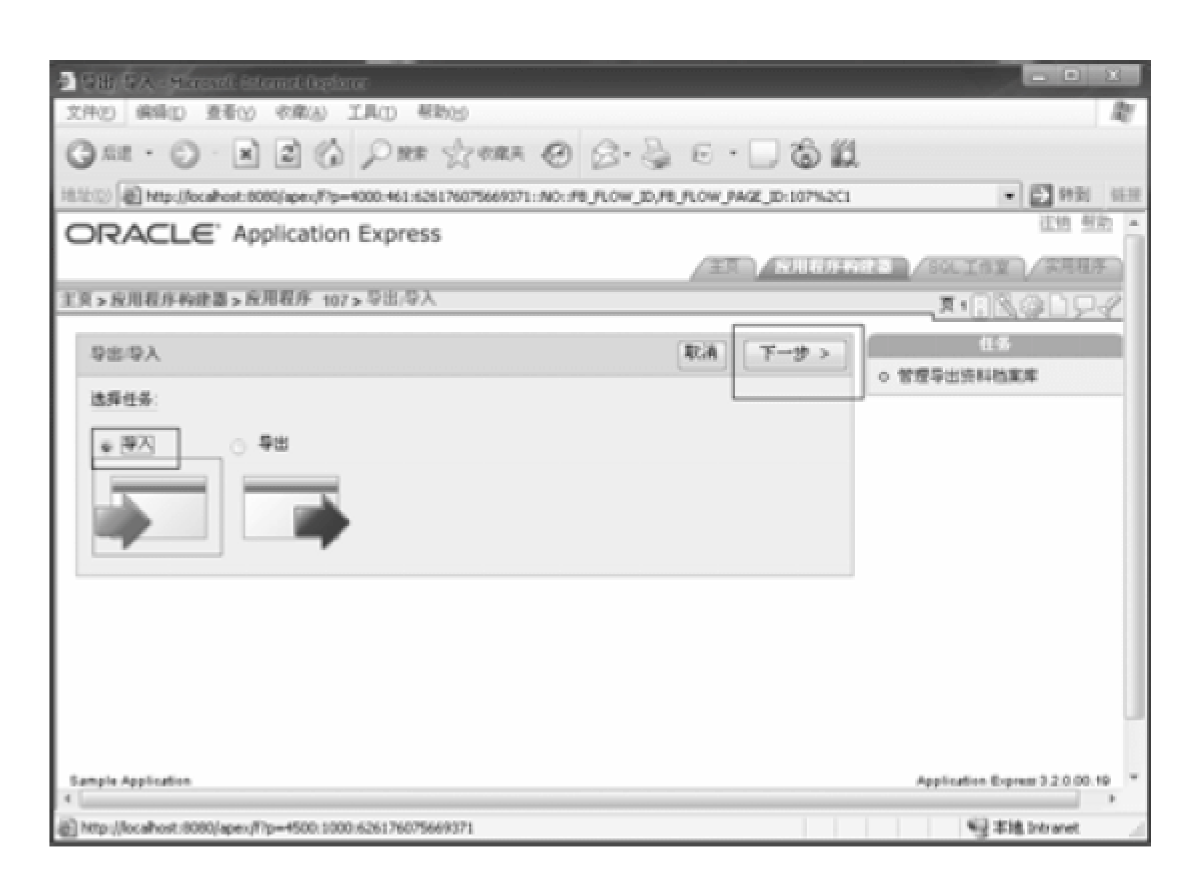


图　23.78

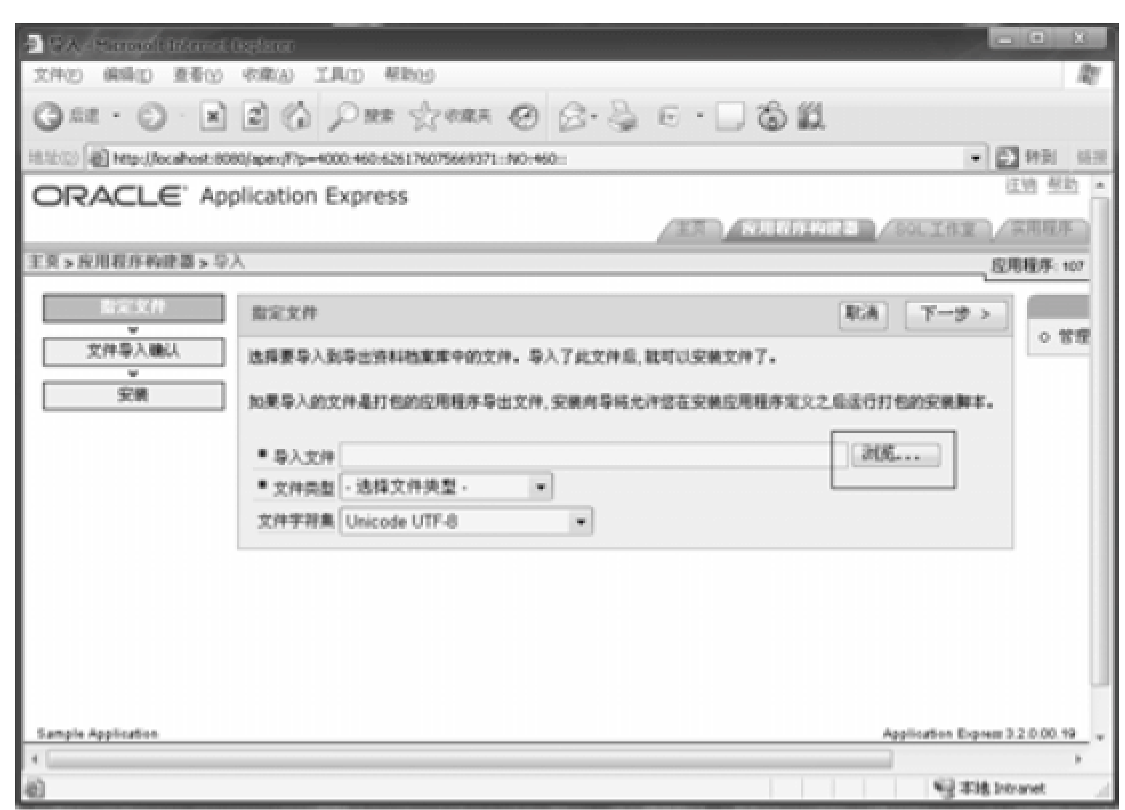


图　23.79

（8）在“查找范围”下拉列表框中选择 F:\Express_Data\customer_tracker_1.1，在“文件名”文本框中输入 customer_tracker_installer_1.1.sql，单击“打开”按钮，如图 23.80 所示。之后将返回“导入”页面。

（9）在“文件类型”下拉列表框中选择“应用程序，页或组件导出”选项，在“文件字符集”下拉列表框中接受默认设置，单击“下一步”按钮，如图 23.81 所示。



图　23.80

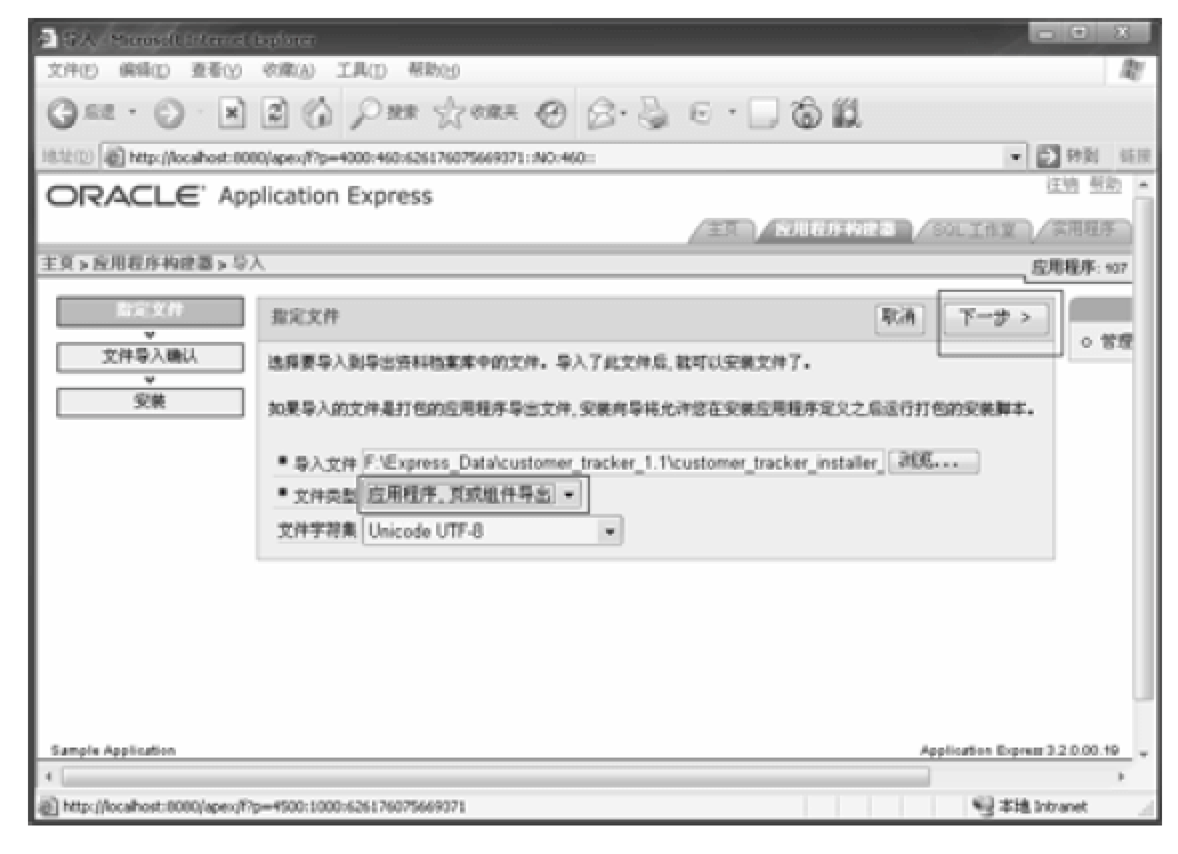


图　23.81

（10）继续单击“下一步”按钮，如图 23.82 所示。之后将进入“安装应用程序”页面。

（11）在“语法分析方案”下拉列表框中选择 CUST，在“构建状态”下拉列表框中选择“运行和构建应用程序”选项，选中“自动分配新应用程序 ID”单选按钮，单击“安装”按钮，如图 23.83 所示。

（12）在“安装支持对象”栏中选中“是”单选按钮，单击“下一步”按钮，如图 23.84 所示。

（13）单击“安装”按钮，完成应用程序软件包的安装，如图 23.85 所示。

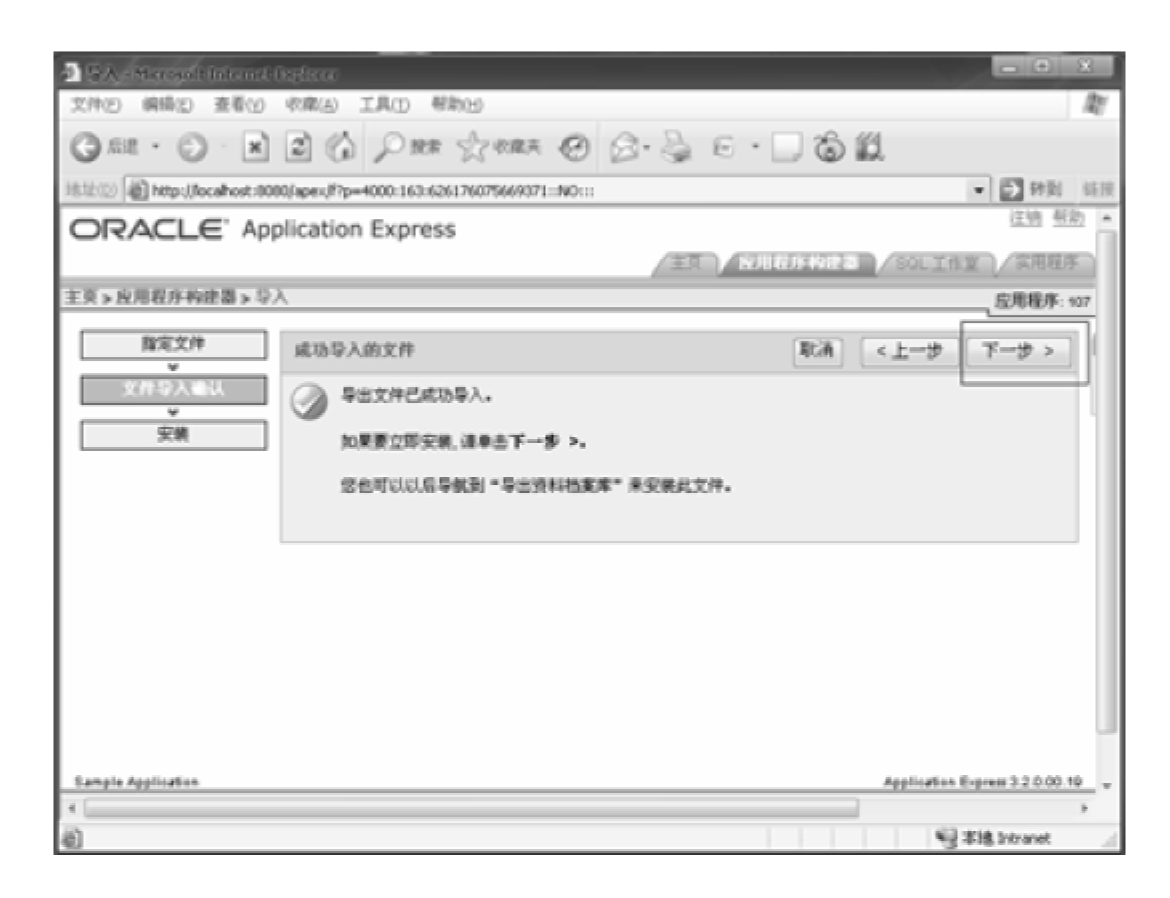


图 23.82

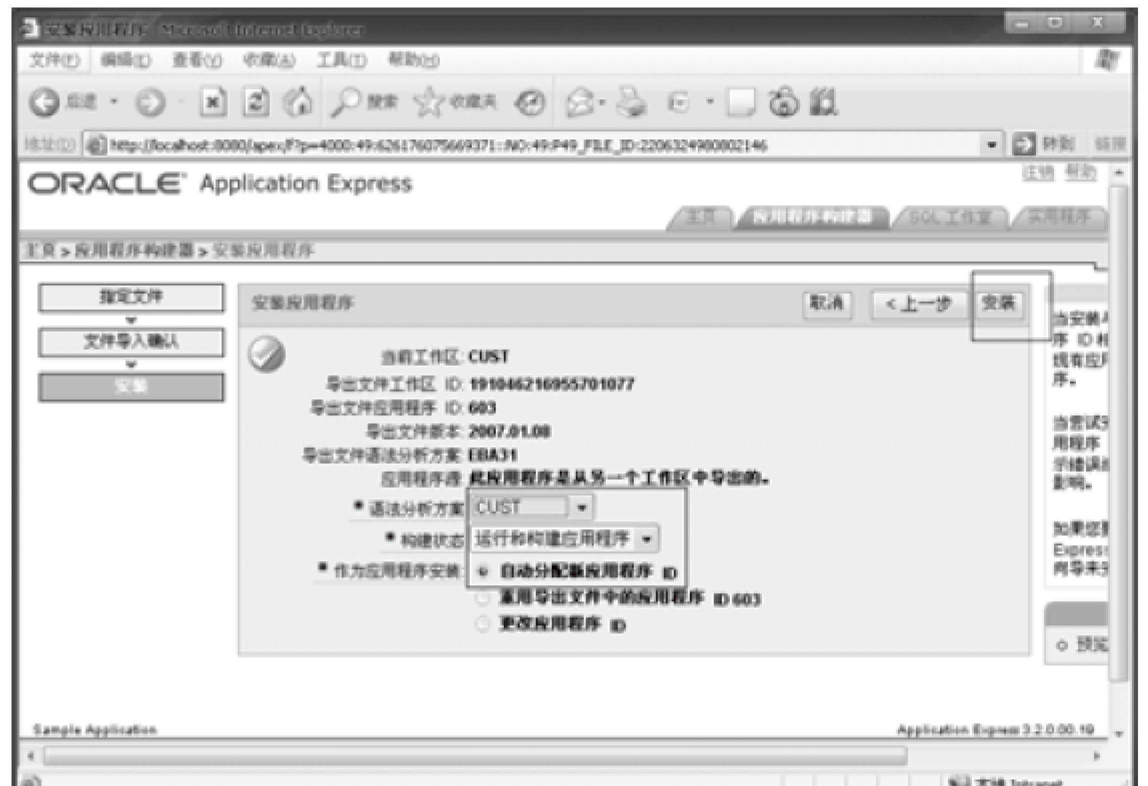


图 23.83



图 23.84

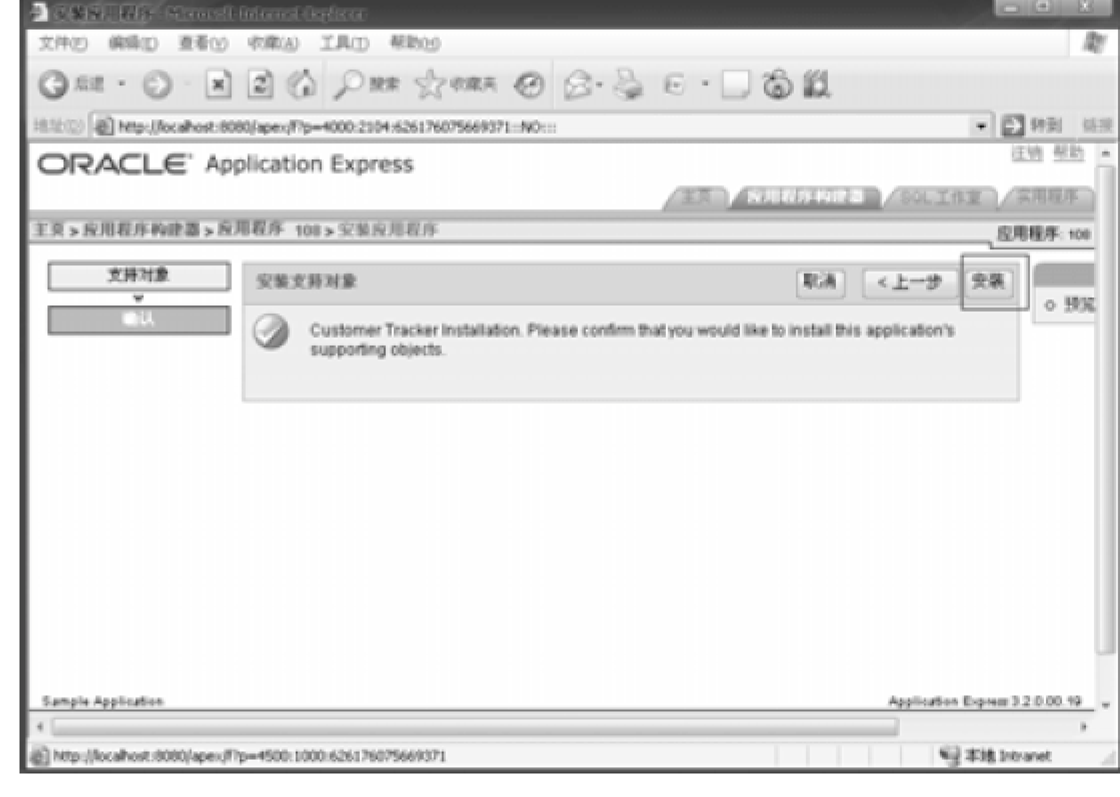


图 23.85

（14）出现如图 23.86 所示的页面就表示您的安装已经成功。此时可以单击“安装概要”图标来浏览安装的概要信息。之后将进入“安装概要”页面。

（15）单击“应用程序构建器”超链接以返回“应用程序构建器”页面，如图 23.87 所示。



图 23.86

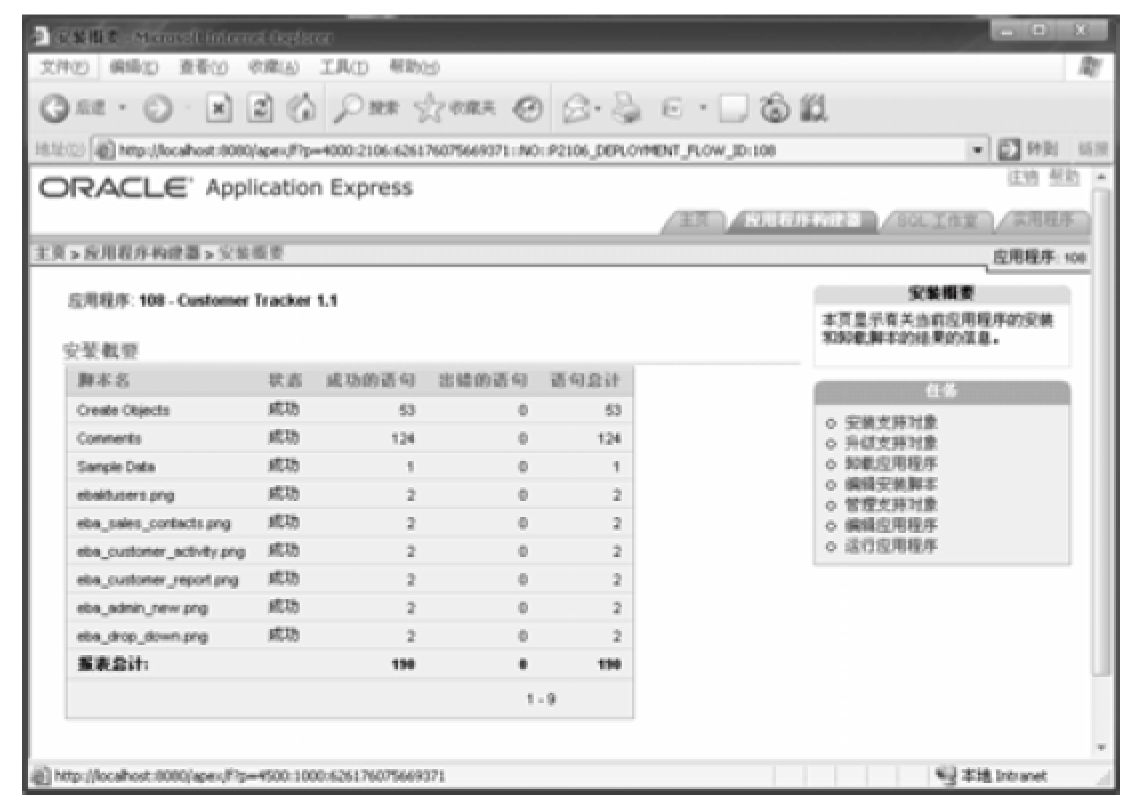


图 23.87

（16）此时，您会发现多了一个名为 Customer Tracker 1.1-108 的应用程序。单击 Customer Tracker 1.1-108 图标，如图 23.88 所示。之后将进入“应用程序 108”页面。

（17）单击“运行应用程序”图标运行该应用程序，如图 23.89 所示。之后将出现 Customer Tracker 应用程序的登录页面。

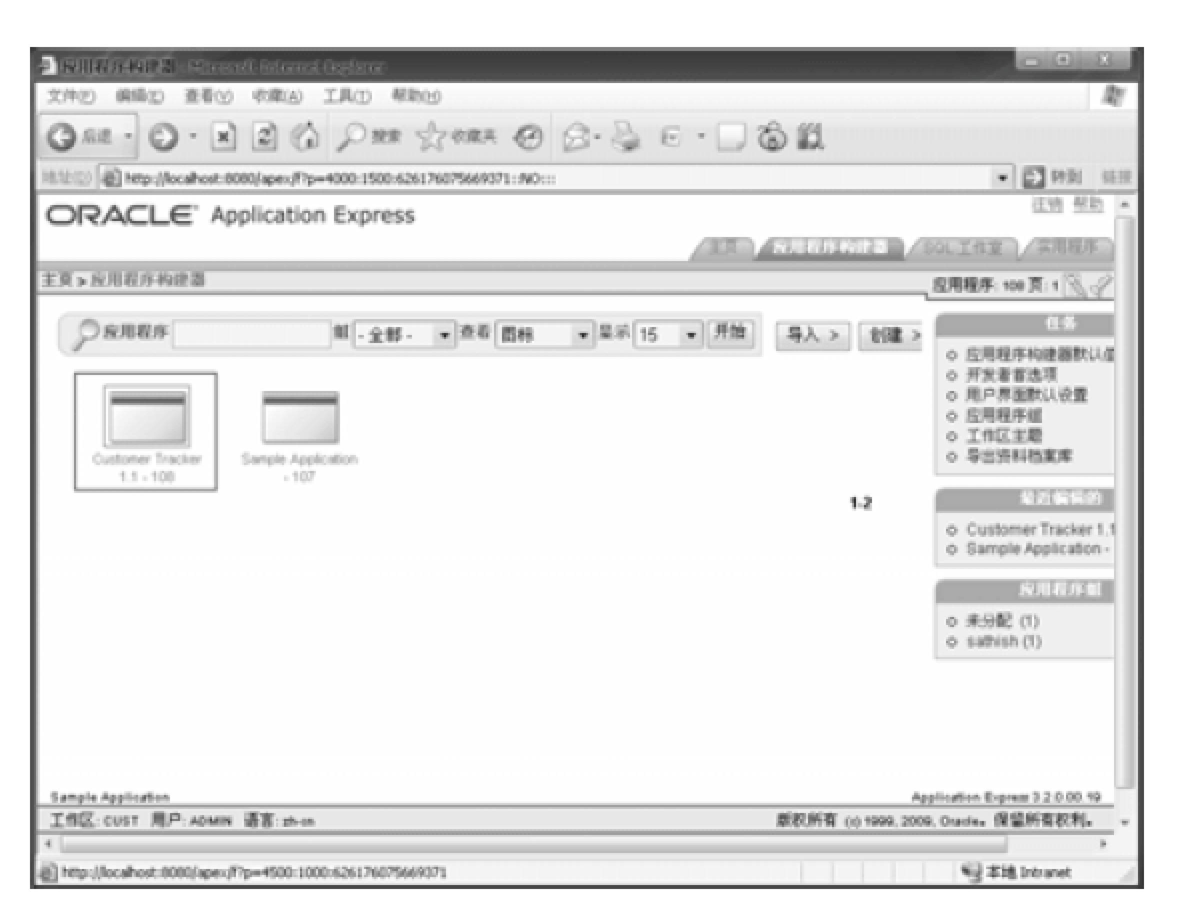


图　23.88



图　23.89

（18）在 User Name 文本框中输入“admin”，在 Password 文本框中输入“xm_Q1ng”，单击 Login 按钮，如图 23.90 所示。

（19）现在您就可以使用这个应用程序了，如单击 Customers 图标，如图 23.91 所示，之后就会出现如图 23.92 所示的客户信息页面。

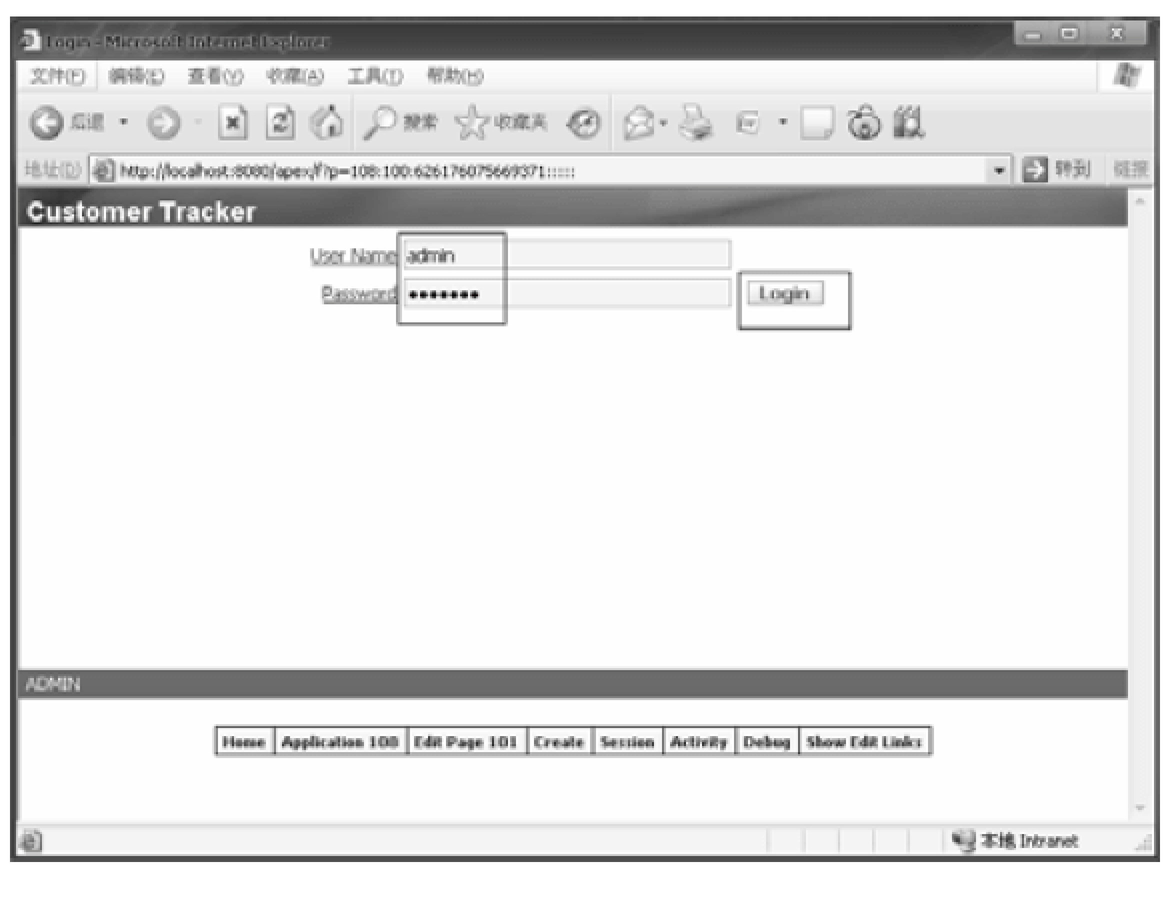


图　23.90



图　23.91

（20）退回到主页（Home）后单击 Dashboard 超链接，如图 23.93 所示，随后就会出现如图 23.94 所示的页面。

可以按照需要选择不同的操作，也可以对这个应用程序进行编辑和扩充以适应您的具体的工作环境。您可以使用类似的方法下载所需的应用程序软件包，然后进行安装并进行编辑和扩充，这样可以极大地加快软件开发的速度，也可以明显地提高软件的质量。因为您是踩

到了许多“大虾”们的肩膀上，所以您就可以爬得更高、升得更快，也可以赚得更多。

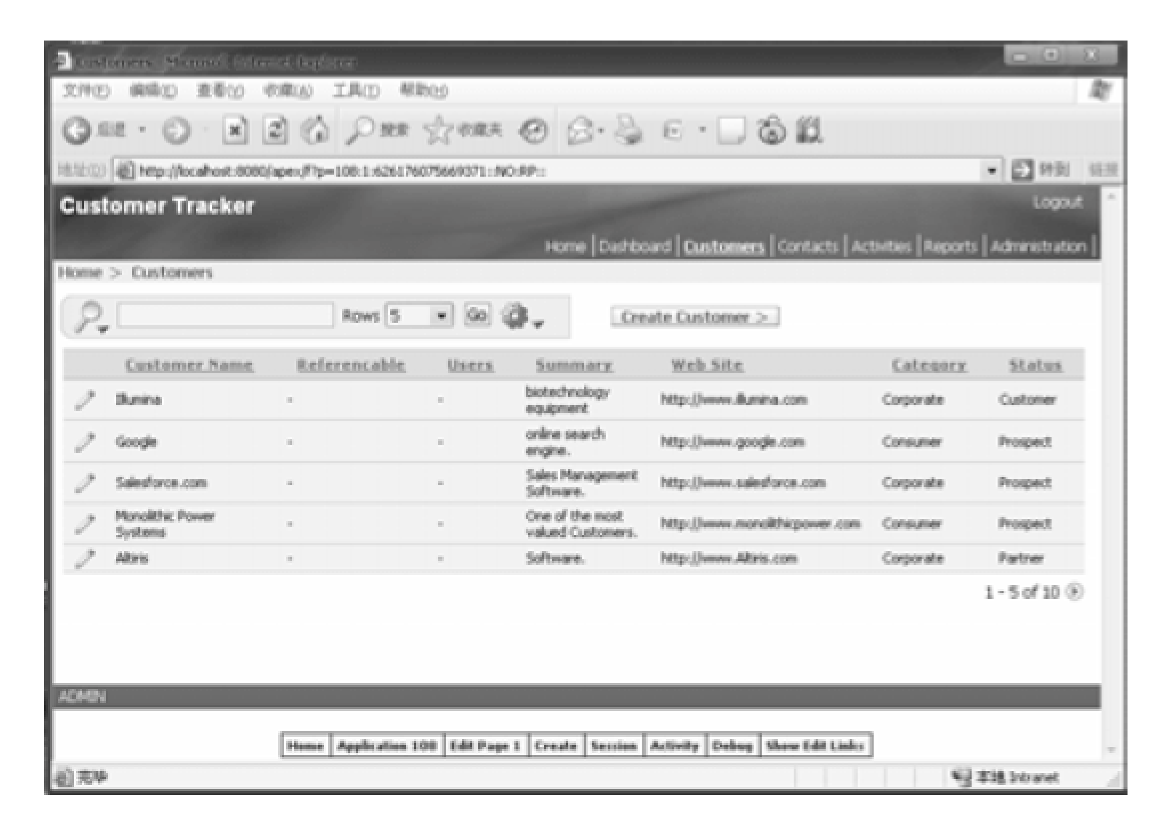


图 23.92



图 23.93

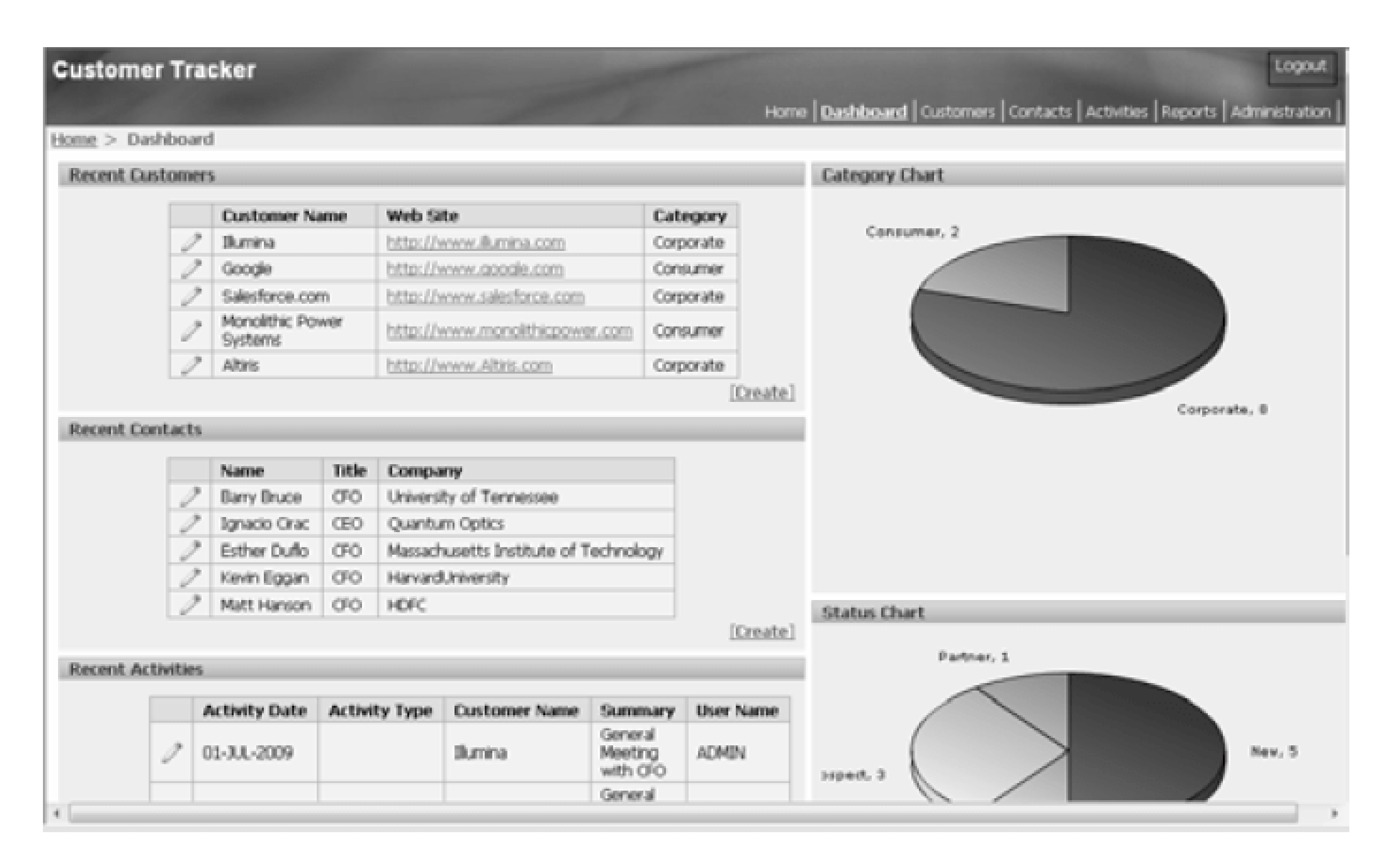


图 23.94

23.10 创建终端用户

当把应用程序移到生产环境之后，您必须为每一个终端用户在 Oracle Application Express 上创建用户账户。为了简单起见，我们将在以下的操作中演示如何在 HR 工作区中创建终端用户。

由于在 HR 工作区上的两个用户权限过大，一个 admin 为管理员，另一个 dog 为开发人员，如果终端用户直接使用他们来访问或操作应用程序将会出现安全问题。于是，您决定为普通用户在 HR 工作区上创建属于他们自己的账户。以下是创建普通用户 BabyDog 的具体操作步骤：

（1）进入 Oracle Application Express 的登录页面。在“工作区”文本框中输入“HR”，在“用户名”文本框中输入“ADMIN”（因为只有管理员用户才能创建新用户），在“口令”文本框中输入“xm_Q1ng”，单击“登录”按钮，如图 23.95 所示。

（2）在右侧的管理列表中单击“管理 Application Express 用户”超链接，如图 23.96 所示。

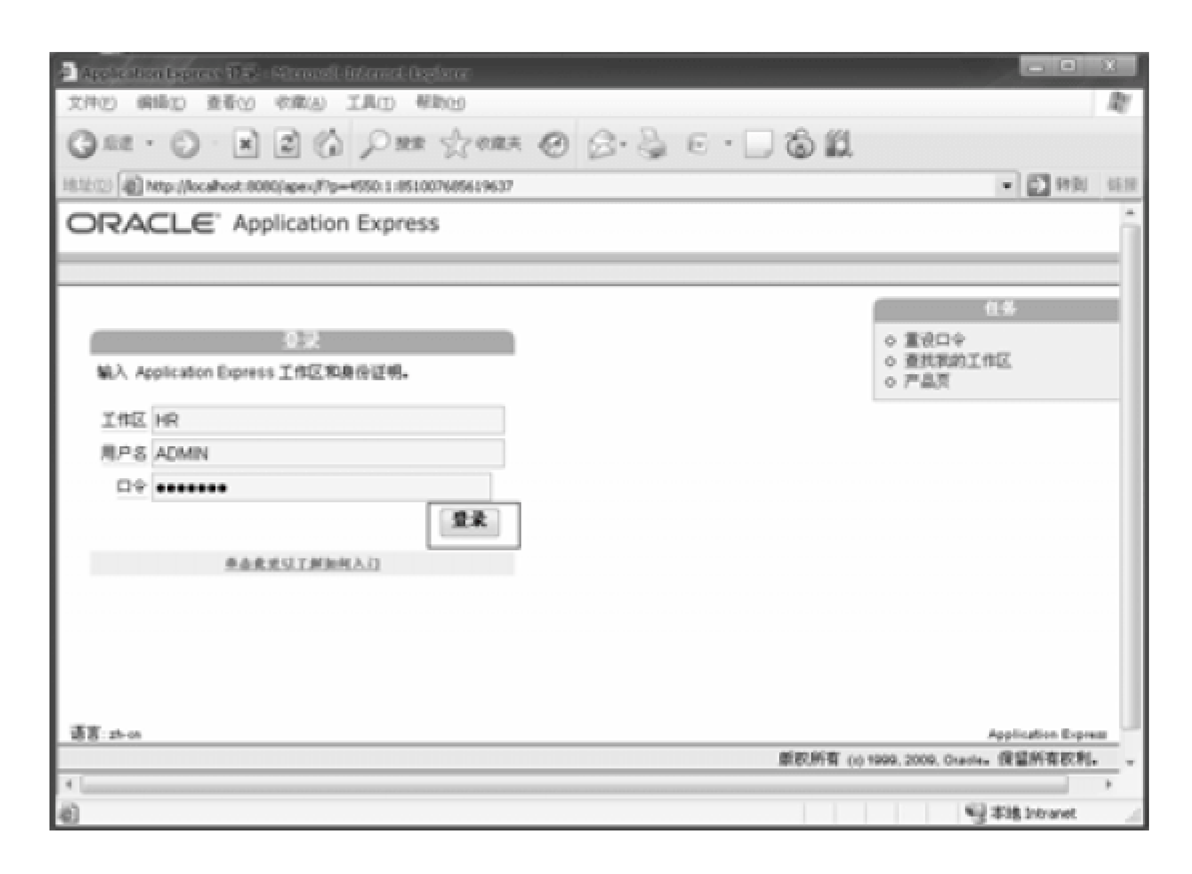


图　23.95

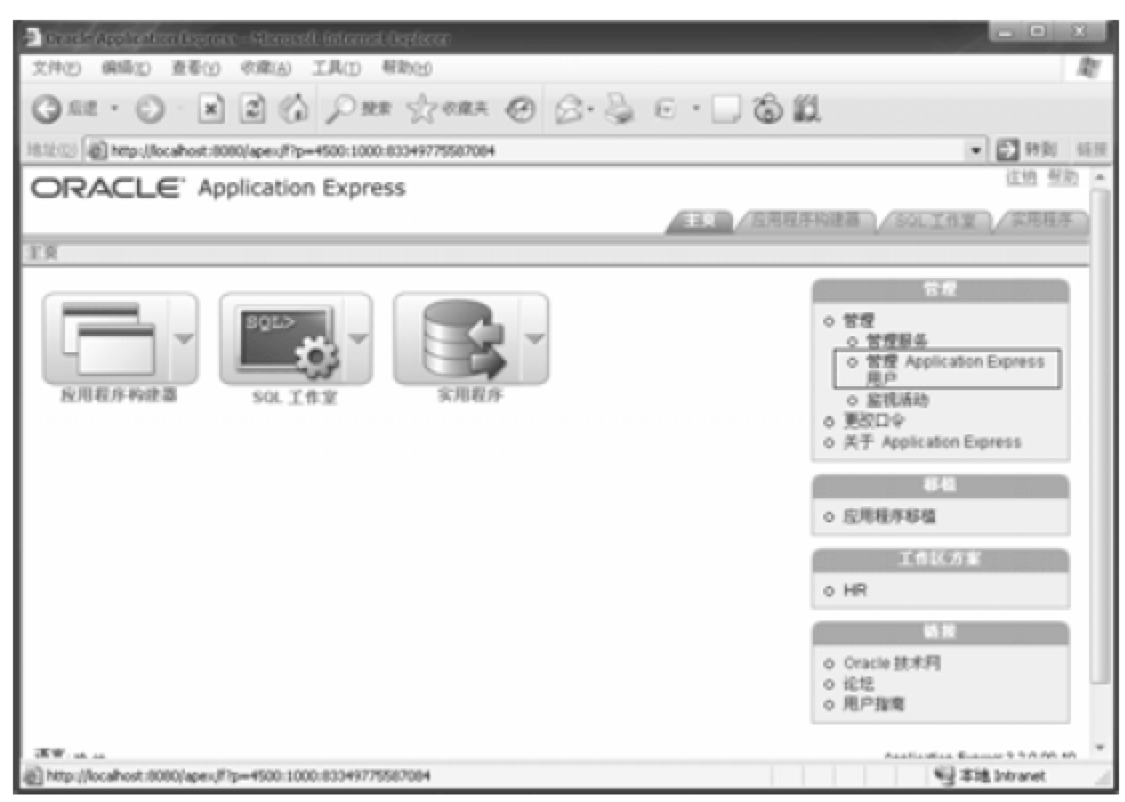


图　23.96

（3）在右下角的任务列表中单击“创建最终用户”超链接，如图 23.97 所示。

（4）打开“创建用户”页面，在“用户名”文本框中输入 BabyDog，在“口令”文本框中输入“W_wan9”，在“确认口令”文本框中继续输入“W_wan9”，在“电子邮件地址”文本框中输入“babydog@baidu.com.cn”（根据实际情况输入），在“默认方案”下拉列表框中选择 HR，在“用户是开发者”栏中选中“否”单选按钮，在“用户是工作区管理员”栏中同样选中“否”单选按钮，在“设置账户可用性”下拉列表框中选择“未锁定”选项，在“需要在首次使用时更改口令”下拉列表框中选择“否”选项，如图 23.98 所示。



图　23.97



图　23.98

☞ **指点迷津：**

出于安全的考虑，用户的口令（密码）应大小写混写，最好至少包含一个字符、一个数字和一个特殊字符。但是这样安全的口令是很难记的，因此有人发明了如下的记忆方法，使用特殊字符替换单词之间的空格；使用数字 0 替代字符 o；使用数字 9 替代字符 q；使用数字 1 替代字符 l 或 I；使用数字 2 替代单词 to；使用数字 4 替代单词 for 等。

（5）在“附加属性”区域，您可以随便输入自己认为需要的信息。此处，在“名”文本框中输入“大郎”，在“姓”文本框中输入“武”，在“说明”文本框中输入“潘金莲的丈夫，武大郎驴肉火烧的鼻祖之一。”，如图 23.99 所示。

（6）向上滚动滚动条，单击“创建用户”按钮，如图 23.100 所示。之后将退回到“管理 Application Express 用户”页面。



图 23.99



图 23.100

（7）您会发现此时已经多了一个新用户，它就是您刚创建的 BABYDOG（小狗）用户，如图 23.101 所示，关闭该页面。

（8）启动网络浏览器并在地址栏处输入“http://localhost:8080/apex”，之后单击“转到”按钮，就将进入 Oracle Application Express 的登录页面。

（9）在“工作区”文本框中输入“HR”，在“用户名”文本框中输入“babydog”，在“口令”文本框中输入“W_wan9”，单击“登录”按钮，如图 23.102 所示。



图 23.101

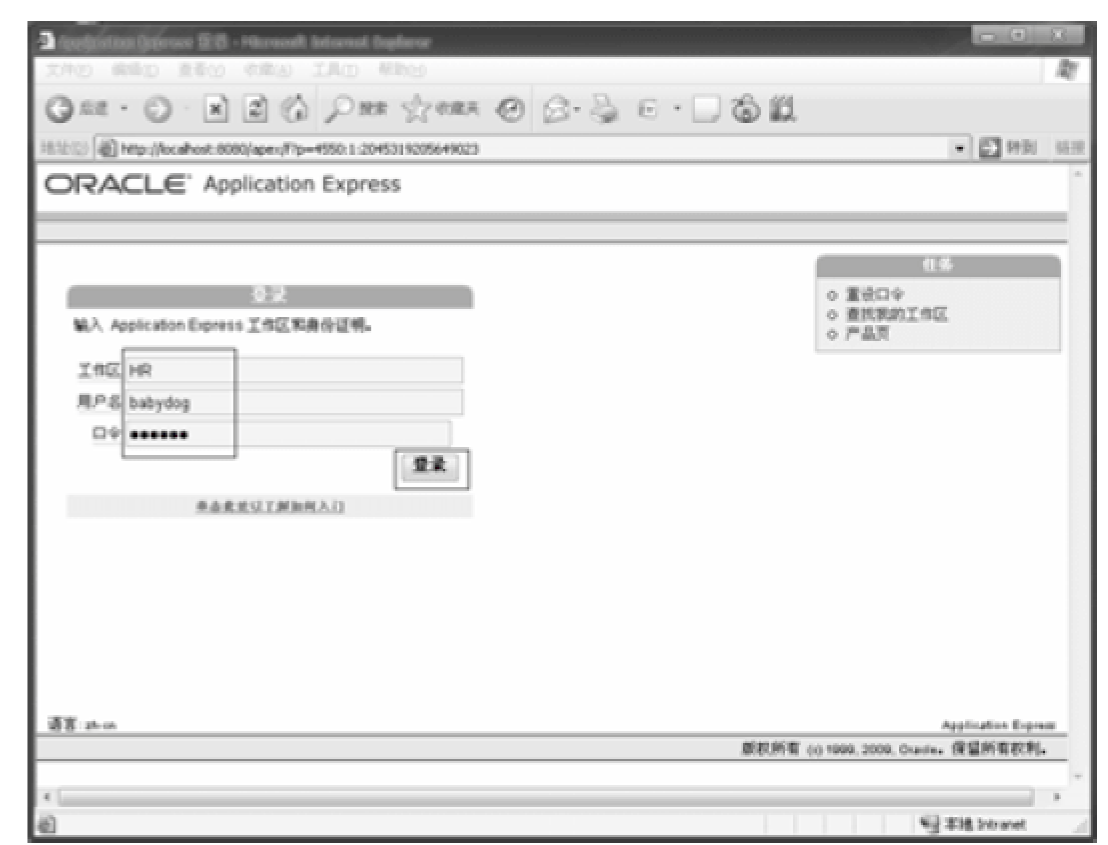


图 23.102

（10）之后将出现如图 23.103 所示的拒绝该用户访问的页面，这表明 babydog 用户确实没有开发人员的权限，最后关闭该页面。

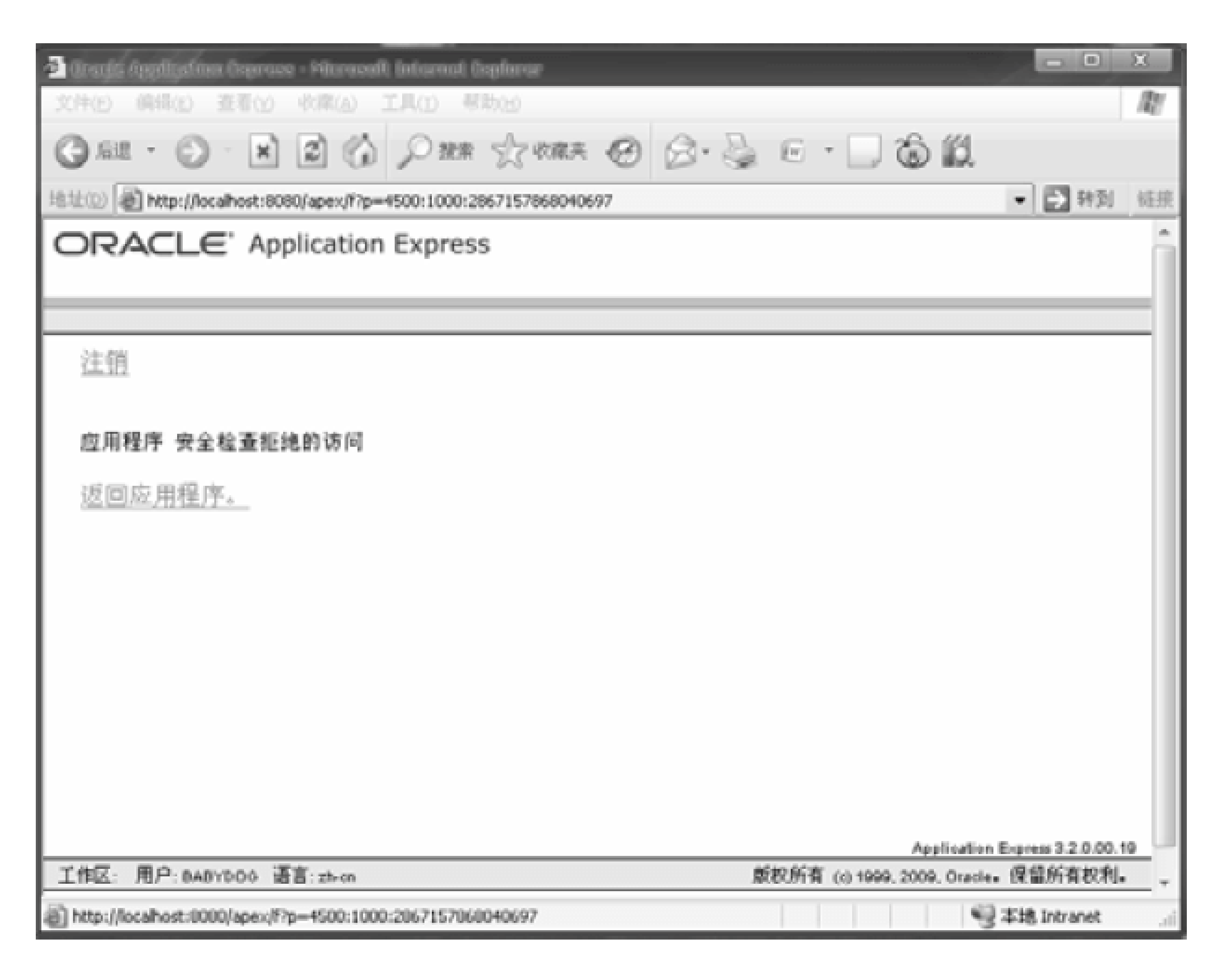


图　23.103

23.11　通过切换主题来改变用户的界面

尽管老板对您开发的应用程序非常满意，但是看久了还是看出点毛病（可能是产生了审美疲劳）。它觉得网页的颜色有些单调，要求您将网页的色彩变得丰富些。于是，您想到了最简单的方法，那就是通过切换主题来改变网页的显示。具体操作步骤如下：

（1）进入 Oracle Application Express 的登录页面并以 DOG 用户身份登录，如图 23.104 所示。之后将进入“主页”页面。

（2）单击“应用程序构建器”图标，如图 23.105 所示。之后将进入“应用程序构建器”页面。

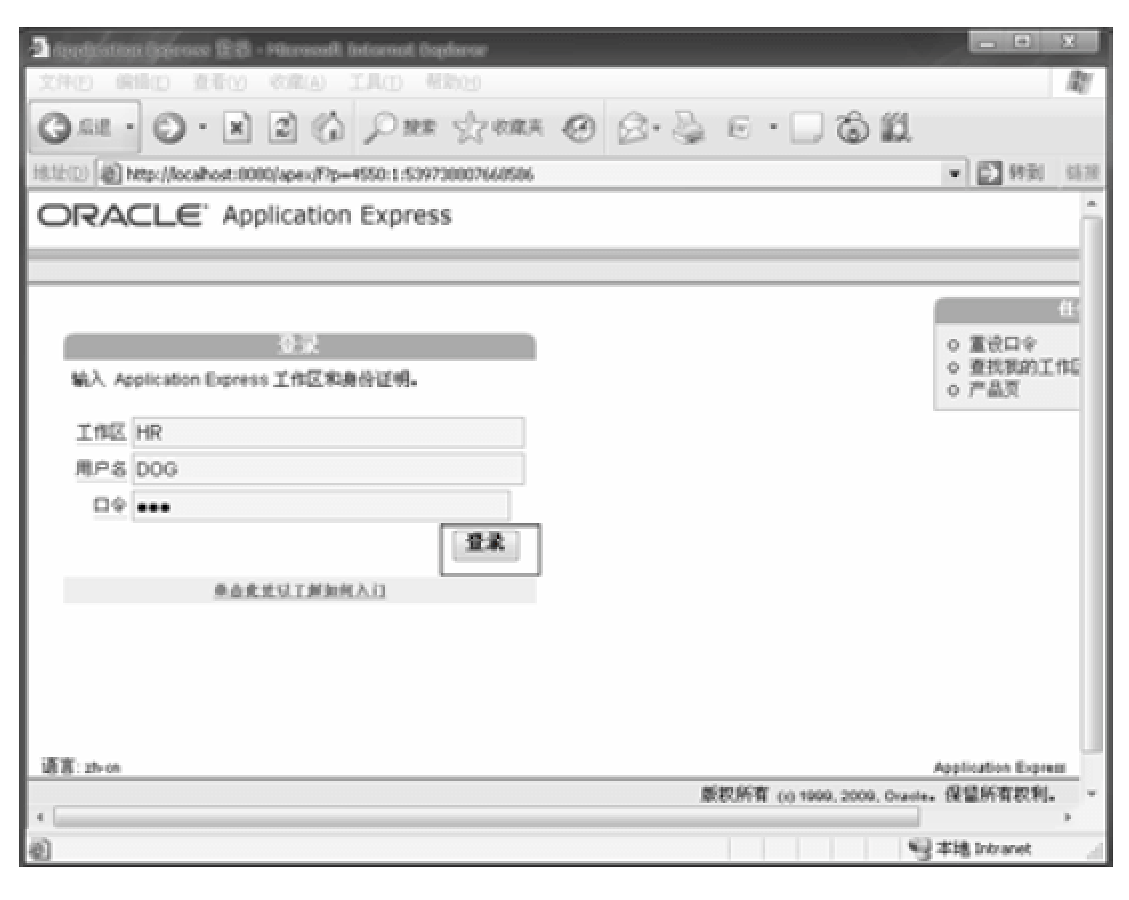


图　23.104

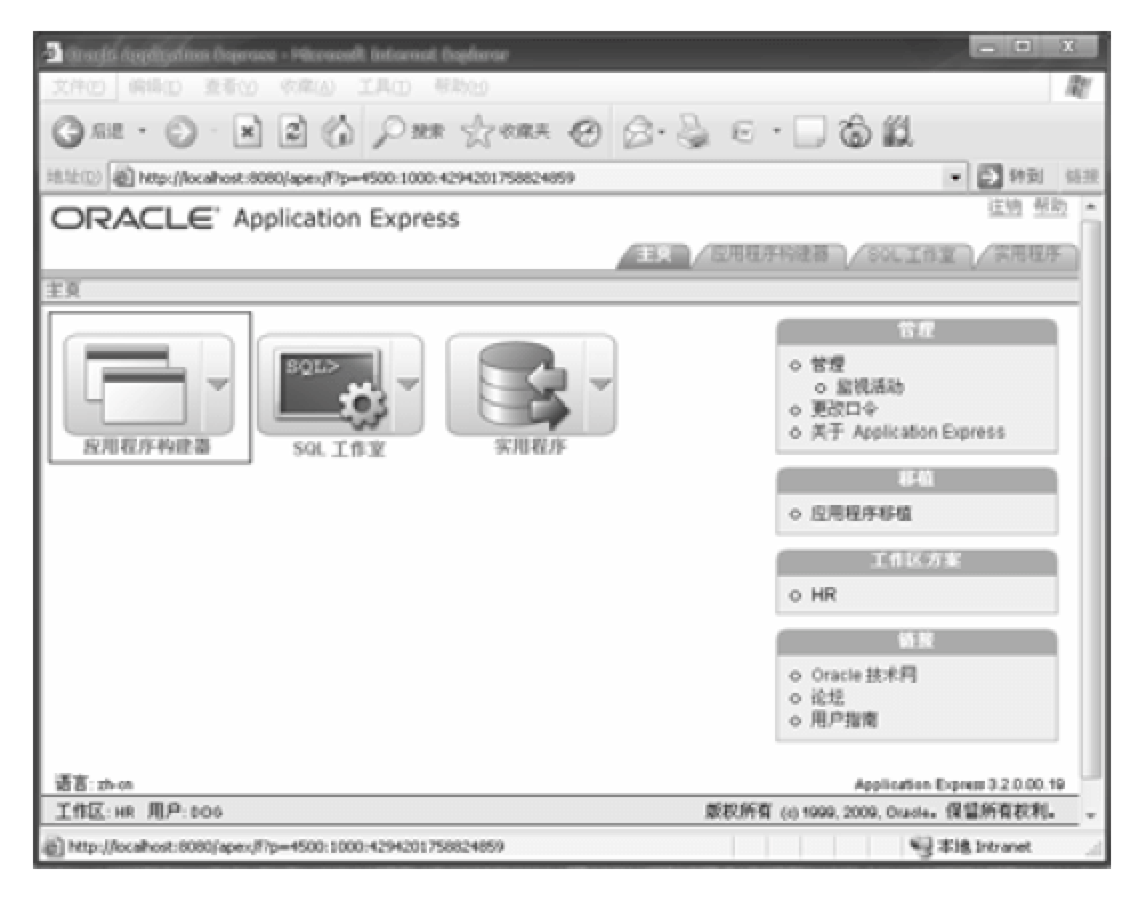


图　23.105

（3）单击 Jinlian 102 图标（在您的系统中可能是 101 或其他的数字，因为我重新创建了 Jinlian 应用程序），如图 23.106 所示。之后将进入“应用程序 102”页面。

（4）单击“共享组件”图标，如图 23.107 所示。之后将进入“共享组件”页面。

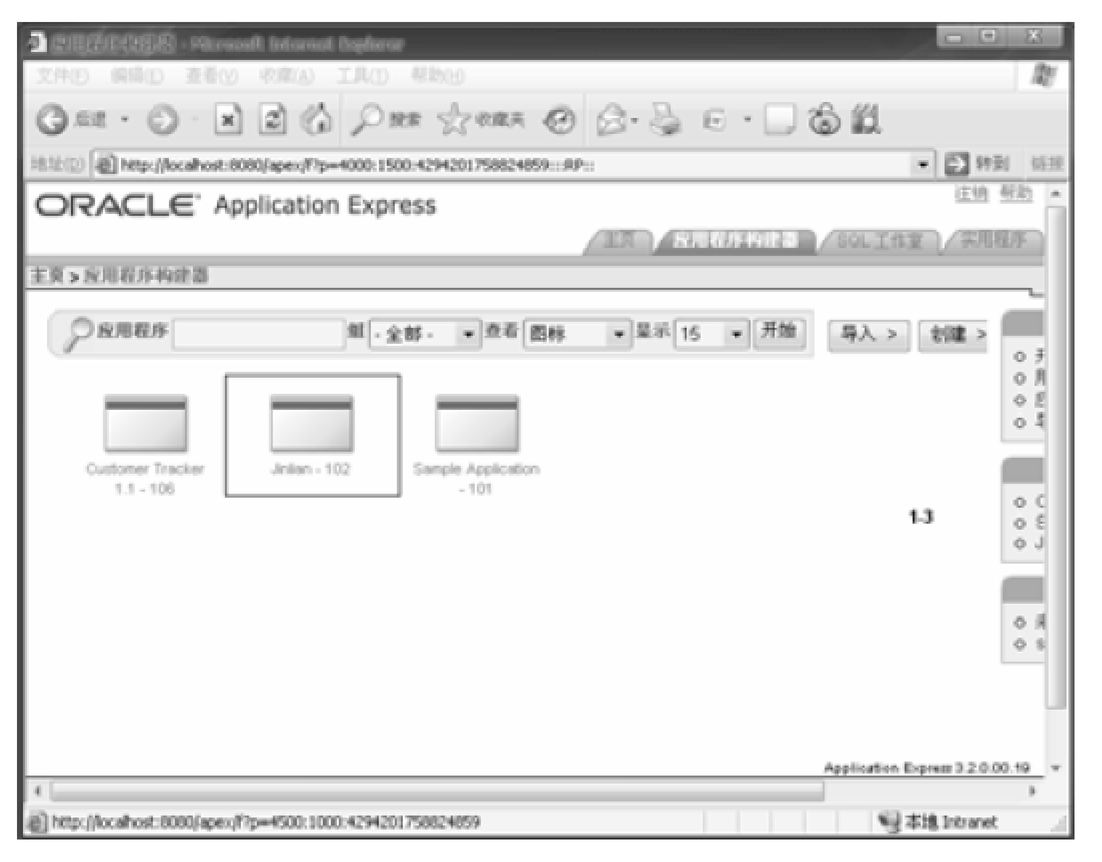


图 23.106



图 23.107

（5）在用户界面之下，单击“主题”超链接，如图 23.108 所示。之后将进入“主题”页面。

（6）单击“创建”按钮，如图 23.109 所示。之后将进入“创建主题”页面。



图 23.108



图 23.109

（7）选中“从资料档案库”单选按钮，单击“下一步”按钮，如图 23.110 所示。

（8）选择您喜欢的主题（这里选择主题 20），如图 23.111 所示。



图 23.110



图 23.111

（9）单击“下一步”按钮，如图 23.112 所示。

（10）单击“创建”按钮，如图 23.113 所示。之后将回到“主题”页面。



图　23.112

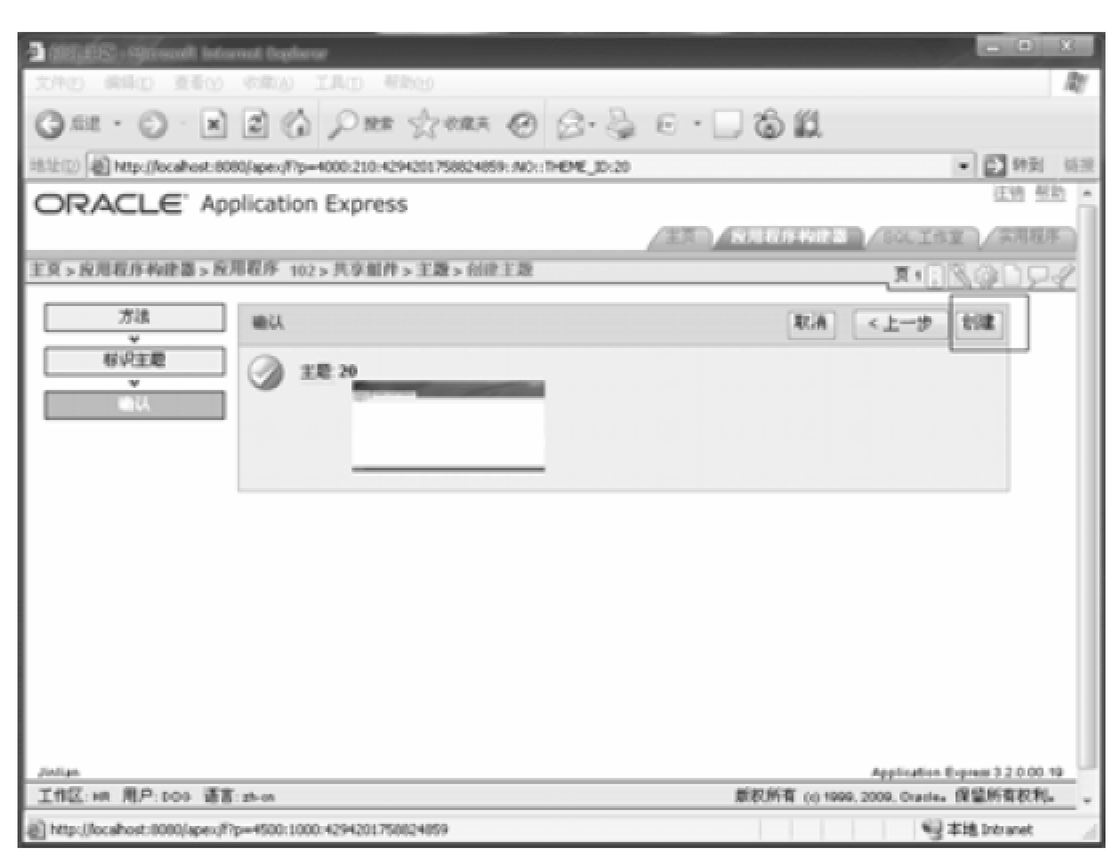


图　23.113

（11）此时您会发现在主题页面中多了一个 Modern Blue-20 主题，这就是您刚创建的主题。单击“切换主题”按钮，如图 23.114 所示，将进入“切换主题”页面。

（12）在“切换到主题”下拉列表框中选择 20. Modern Blue 选项，单击“下一步”按钮，如图 23.115 所示。



图　23.114

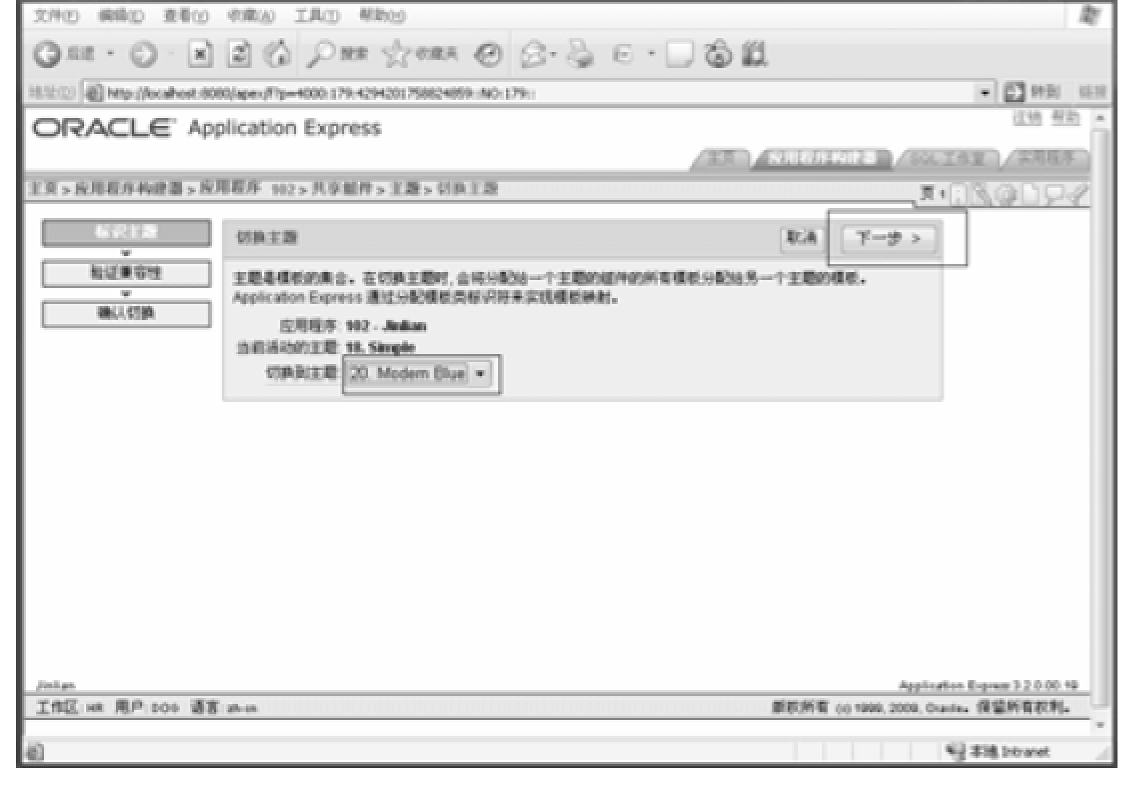


图　23.115

（13）检查所显示的信息是否正确。如果不正确，则退回去重新修改；如果正确，则单击“下一步”按钮，如图 23.116 所示。

（14）单击“切换主题”按钮，如图 23.117 所示。之后将回到“主题”页面。

（15）单击右上角的“运行页”图标，如图 23.118 所示。

之后就会得到如图 23.119 所示的页面，色彩的确要比之前丰富多了，但是否美观却是“仁者见仁、智者见智”。



图 23.116

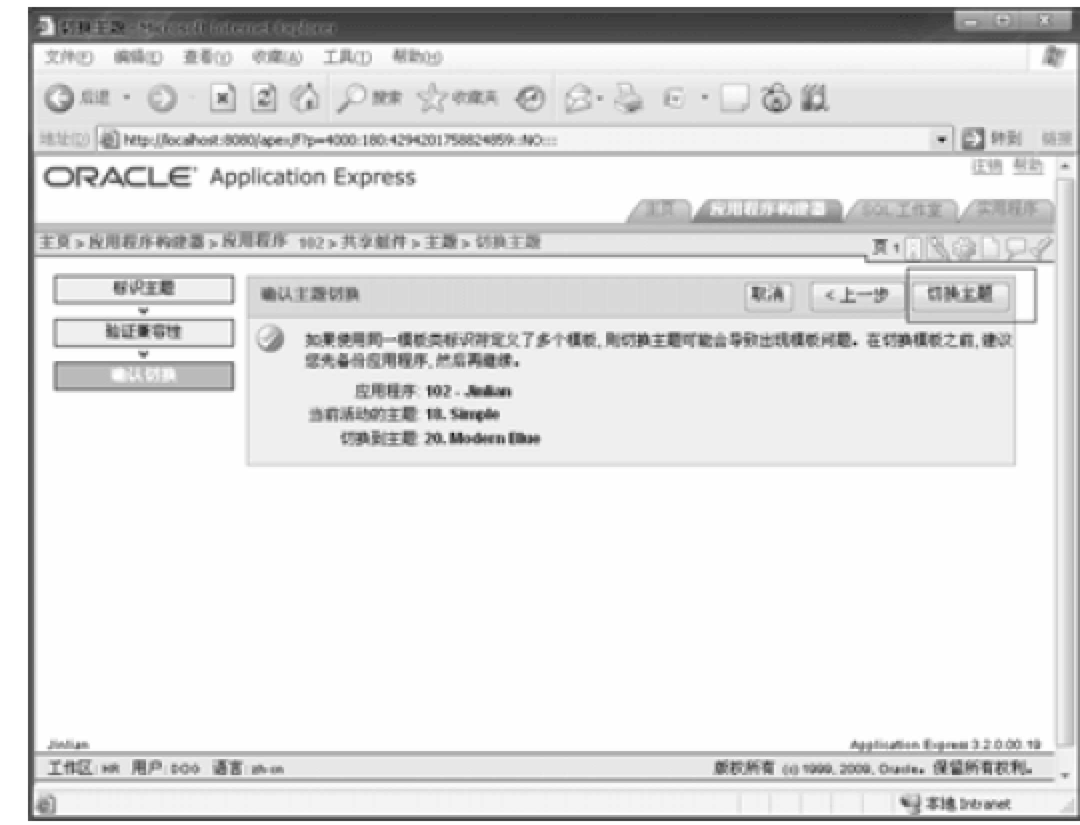


图 23.117





图 23.118

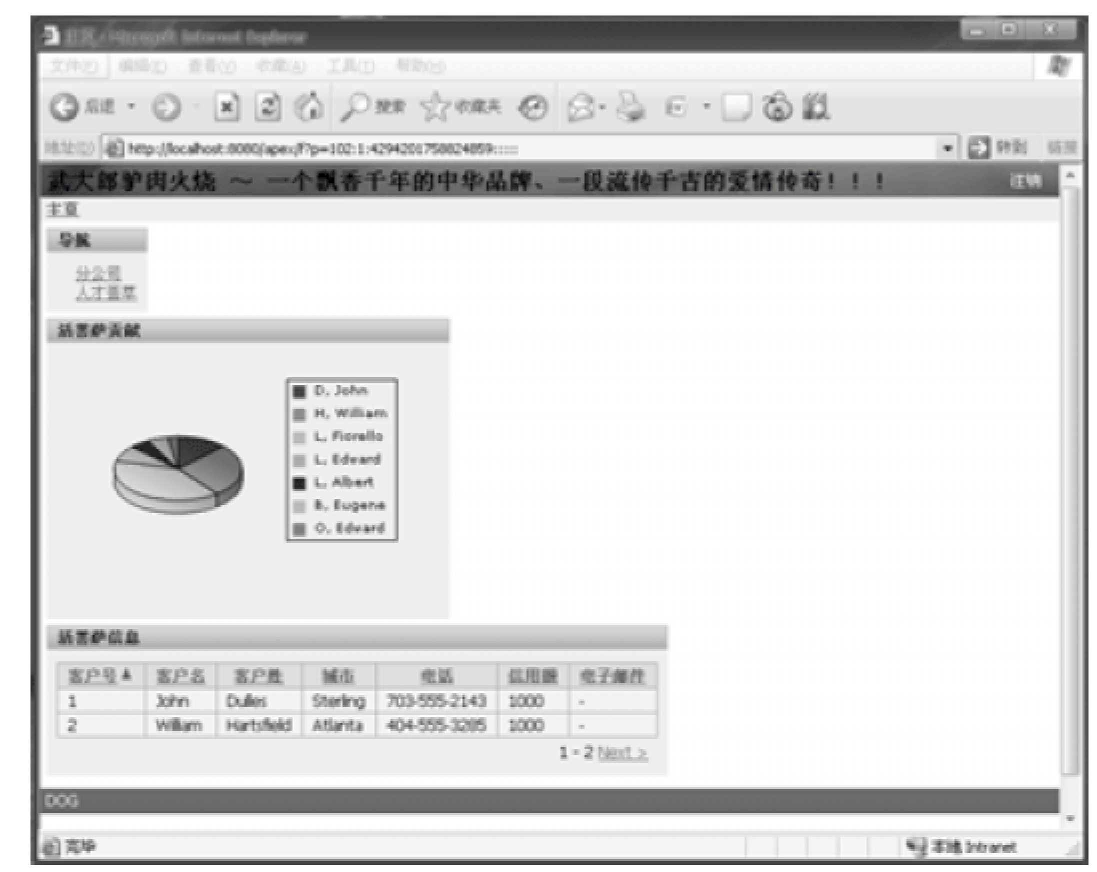


图 23.119

如果老板是一个经常会有审美疲劳的人，隔一段时间她就觉得网页的色彩开始单调了，接下来就要求您进行修改，那又该如何应付呢？其实办法也很简单，就是按以上的方法为您的应用程序创建多个主题，等老板开始感到审美疲劳时，您切换一下主题就行了。这样做您并未改变应用程序的逻辑而只是改变了页面的外观显示，而且老板和用户经常看到不同的网页显示总是有一种新鲜感。

23.12 公布应用程序的网址

之前我们使用最终用户 babydog 试图登录开发人员使用的界面，但是发现该用户无法访问所创建的应用程序。要使普通（终端）用户可以访问这个应用程序，就必须先公布应用程序的网址（URL）。之后普通（终端）用户就可以使用这个网址来访问该应用程序。以下就是公布 jinlian 应用程序的 URL 的具体操作步骤：

（1）启动网络浏览器并转到 Oracle Application Express 的登录页面，随即以 DOG 用户身份登录，如图 23.120 所示。之后将进入 Express 的“主页”页面。

（2）单击“应用程序构建器”图标，如图 23.121 所示。之后将进入 Express 的“应用程序构建器”页面。

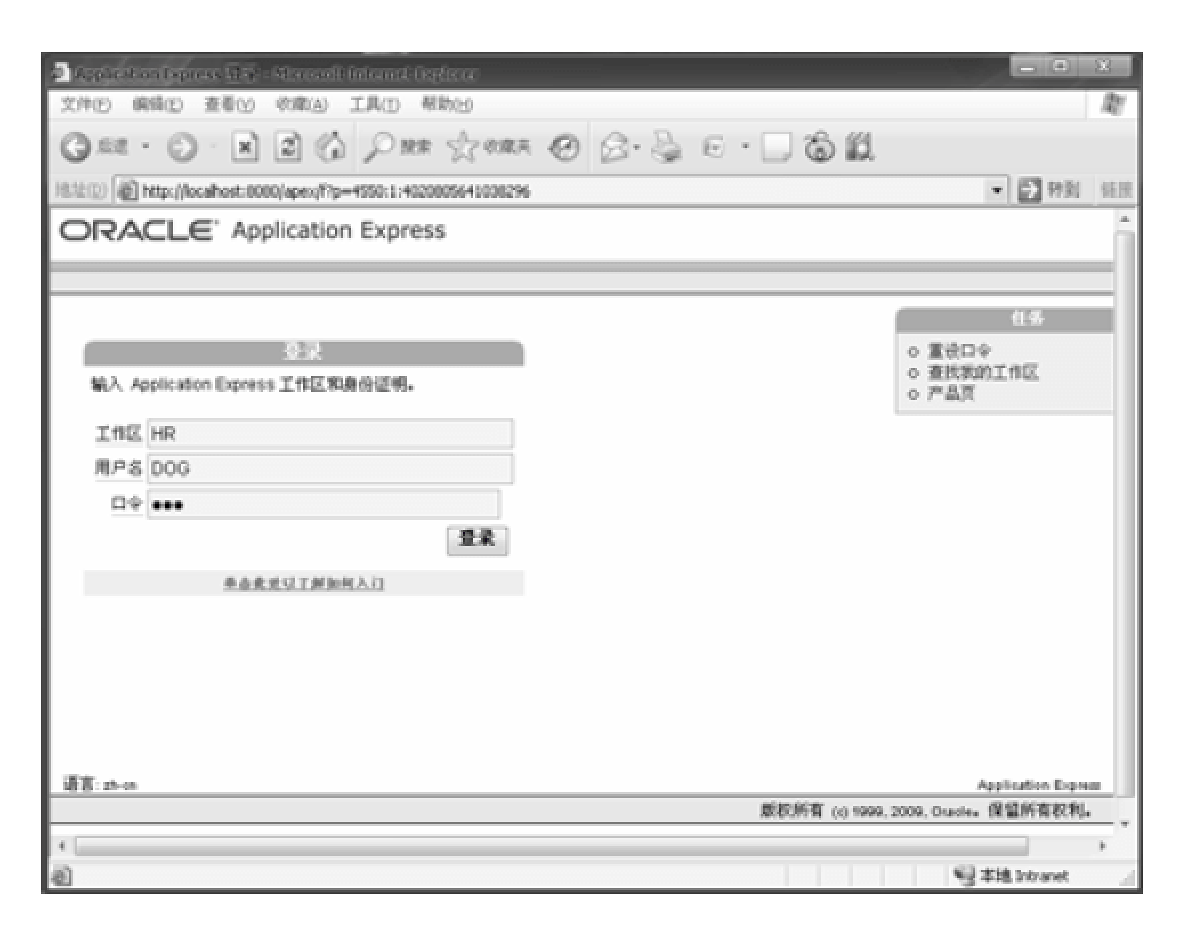


图　23.120

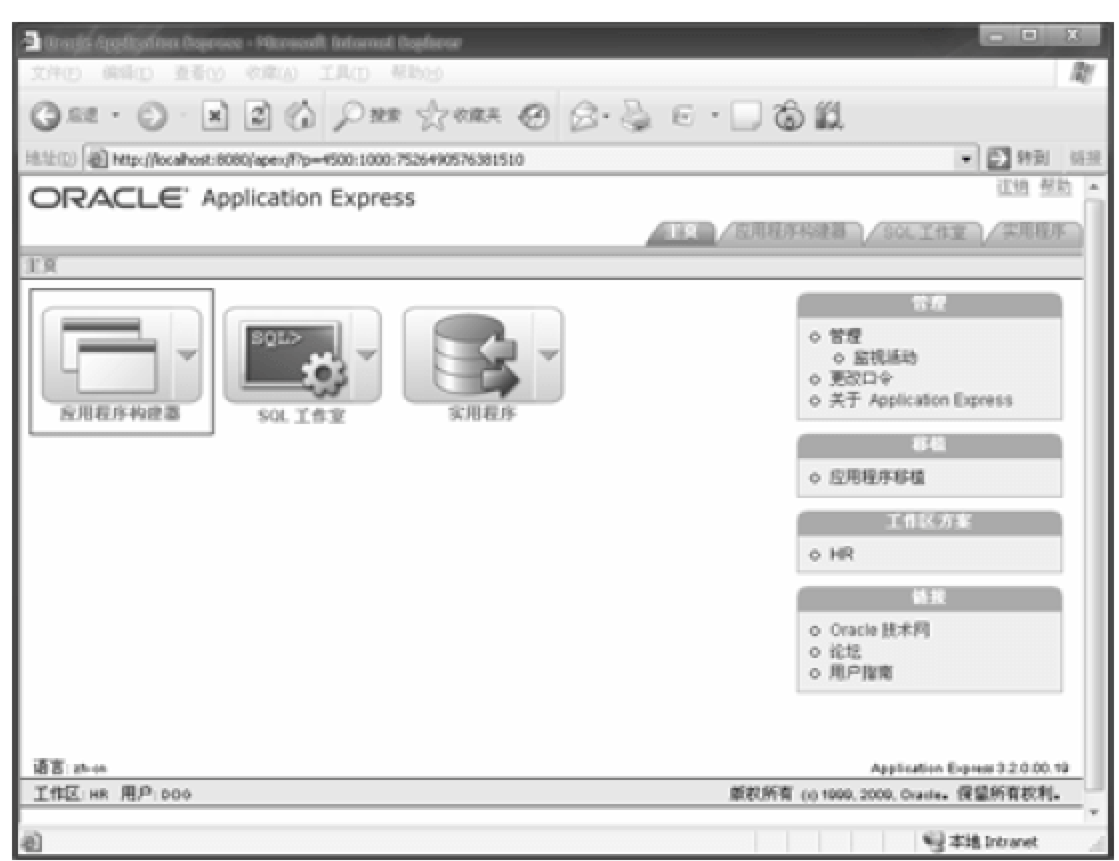


图　23.121

（3）单击 Jinlian-102 图标，如图 23.122 所示。之后将进入 Jinlian-102 页面。

（4）单击“运行应用程序”图标，如图 23.123 所示。之后会出现 Jinlian 应用程序的登录页面。

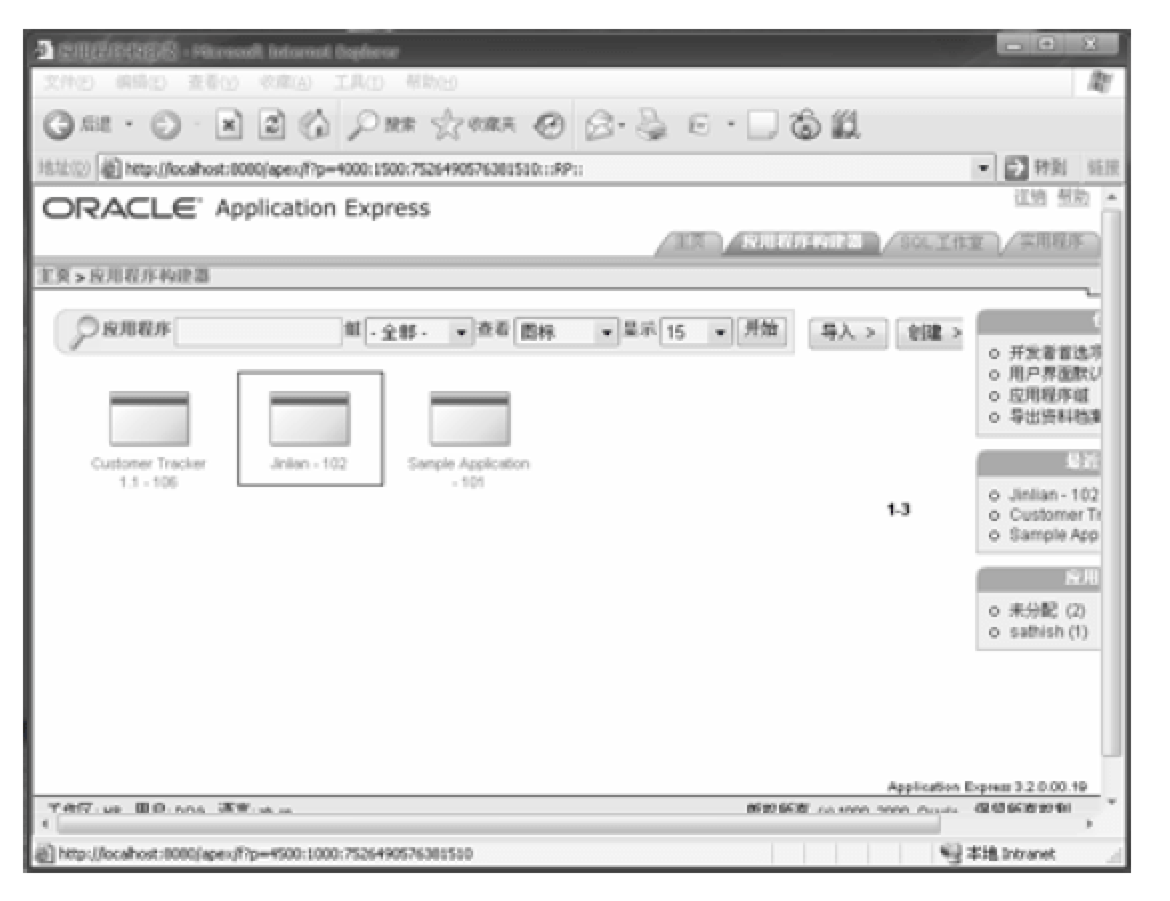


图　23.122



图　23.123

（5）以 dog 用户身份登录该应用程序，如图 23.124 所示。之后将进入该应用程序的主页。

（6）复制地址栏中的网址的 http://localhost:8080/apex/f?p=102:1 部分（注意，不要包括最后面的:979686956501242:::::），如图 23.125 所示。可以将这个网址存入一个记事本文件中以方便以后的使用。

（7）退出该应用程序，如图 23.126 所示。

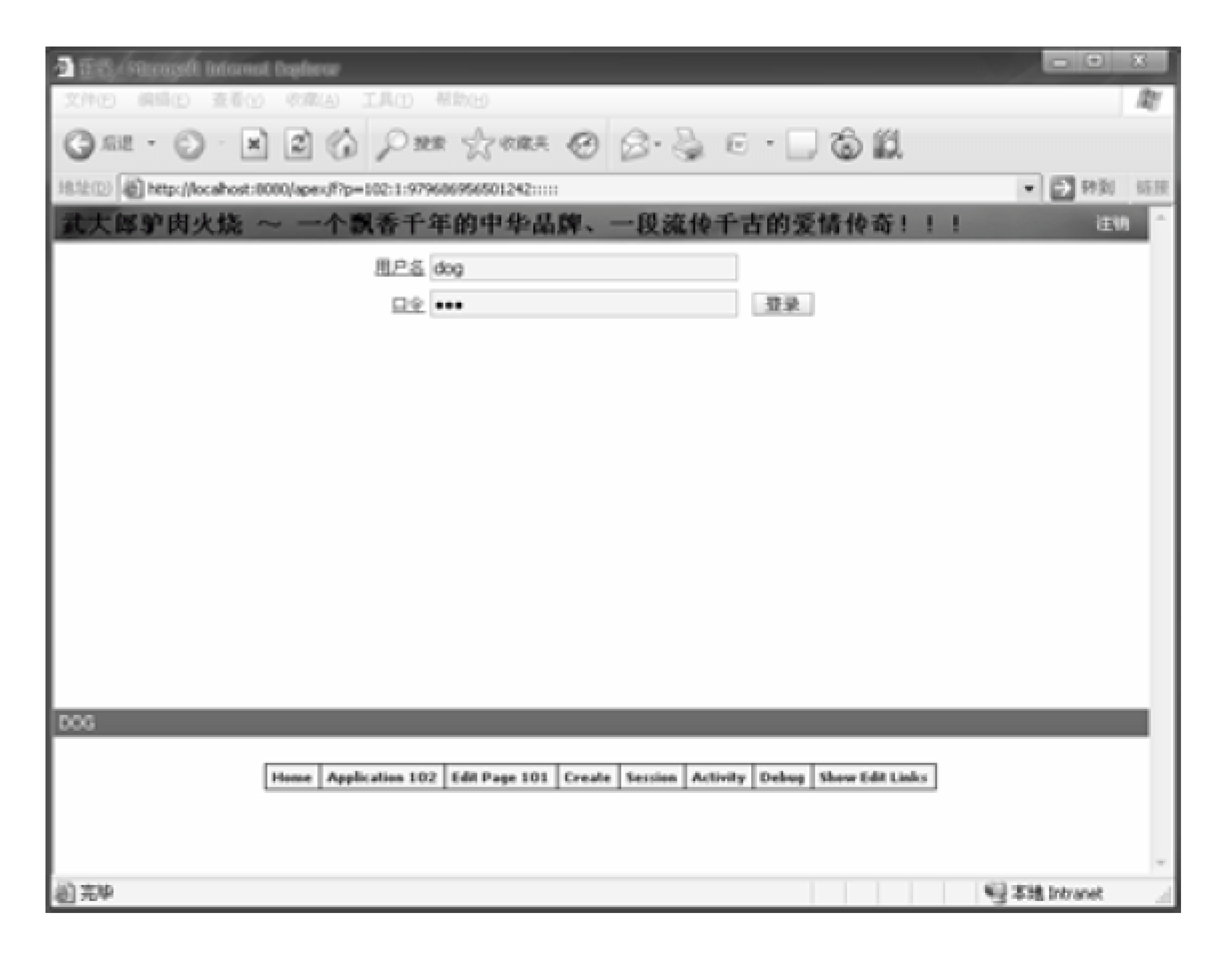


图　23.124



图　23.125



图　23.126

通过以上的操作就完成了 jinlian 应用程序 URL 的公布，下面我们将详细地解释地址栏中地址 http://localhost:8080/apex/f?p=102:1:979686956501242 每一部分的具体含义，其中：

- localhost:8080 是服务器的 URL（网址）。
- apex 是数据库描述符（Database Access Descriptor，DAD）的名字。数据库描述符描述 Oracle HTTP 服务器与数据库服务器的连接方式，以便数据库服务器能够执行 HTTP 的请求。数据库描述符的默认值就是 apex。
- f?p=是 Oracle Application Express 所使用的一个前缀。
- 102 是被调用的应用程序（ID）。
- 1 是应用程序中要显示的页。
- 979686956501242 是会话号（ID），Express 为每个用户对应用程序的每次访问产生唯一的一个会话号，Express 利用会话 ID 来隐含地维护会话的状态。

23.13　普通用户利用公布的URL访问应用程序

既然已经获得了 Jinlian 应用程序的 URL，接下来任何 Express 的合法用户都可以访问这个应用程序了。普通用户利用公布的 URL 访问应用程序的具体步骤如下：

（1）启动网络浏览器，在地址栏处输入“http://localhost:8080/apex/f?p=102:1”，单击“转到”按钮，如图 23.127 所示。随后将进入 Jinlian 应用程序的登录页面。

（2）在“用户名”文本框中输入“babydog”，在“口令”文本框中输入“W_wan9”，单击“登录”按钮，如图 23.128 所示。之后将进入应用程序的“主页”页面。



图　23.127

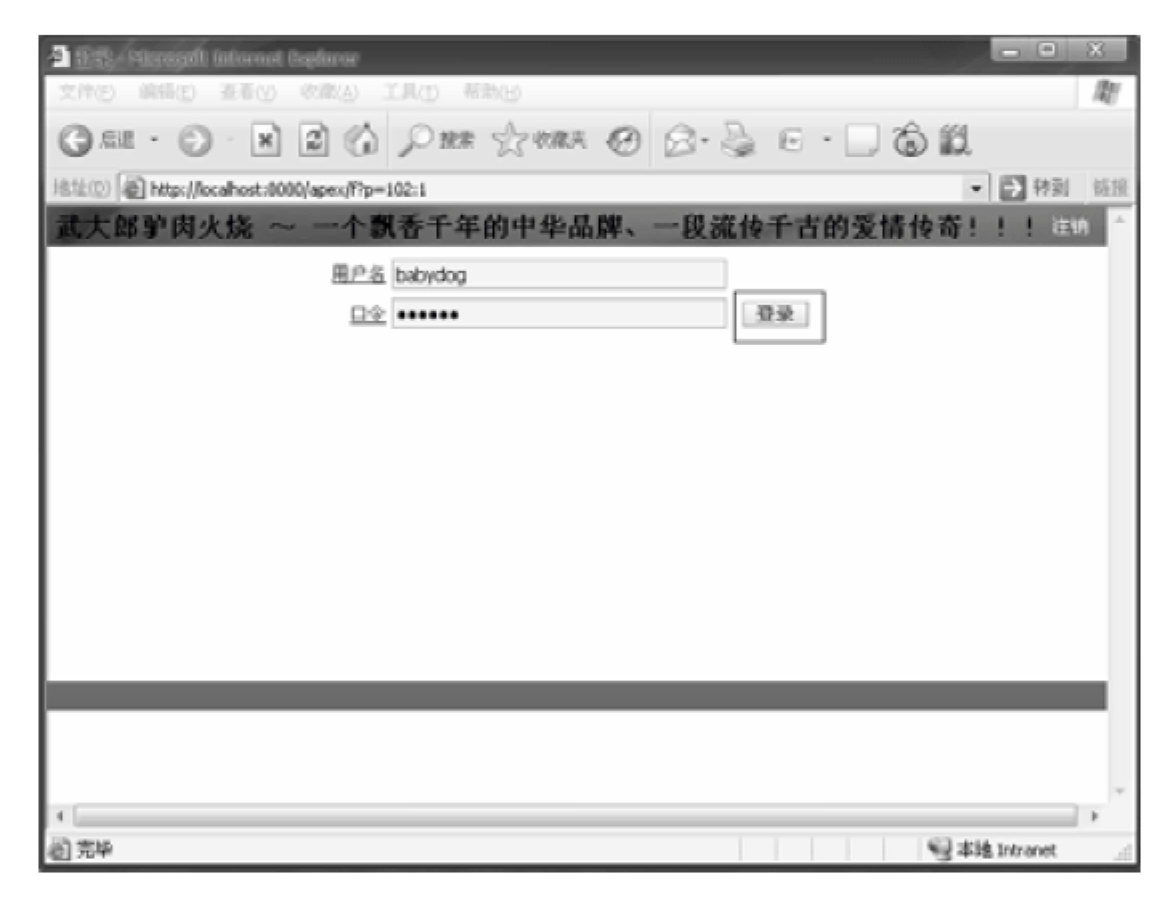


图　23.128

（3）单击导航下的“分公司”超链接，如图 23.129 所示。之后将进入“分公司”页面。

（4）此时，用户就可以浏览分公司的信息，浏览后可以单击“后退”按钮以退回到前一页面，如图 23.130 所示。



图　23.129

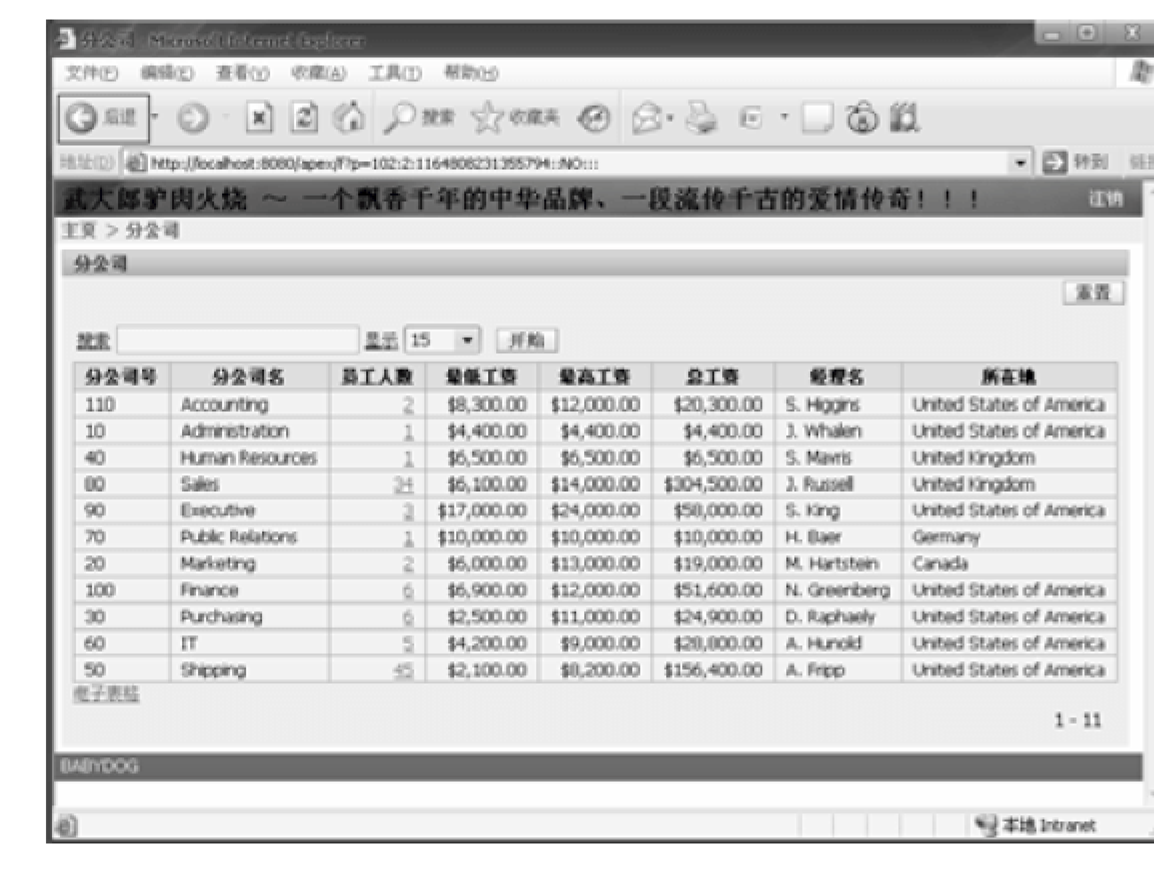


图　23.130

（5）单击导航下的“人才荟萃”超链接，如图 23.131 所示。之后将进入“人才荟萃”页面。

（6）此时，用户可以同时浏览分公司的信息和该分公司中员工的信息，如图 23.132 所示。选择分公司，如 IT。



图　23.131

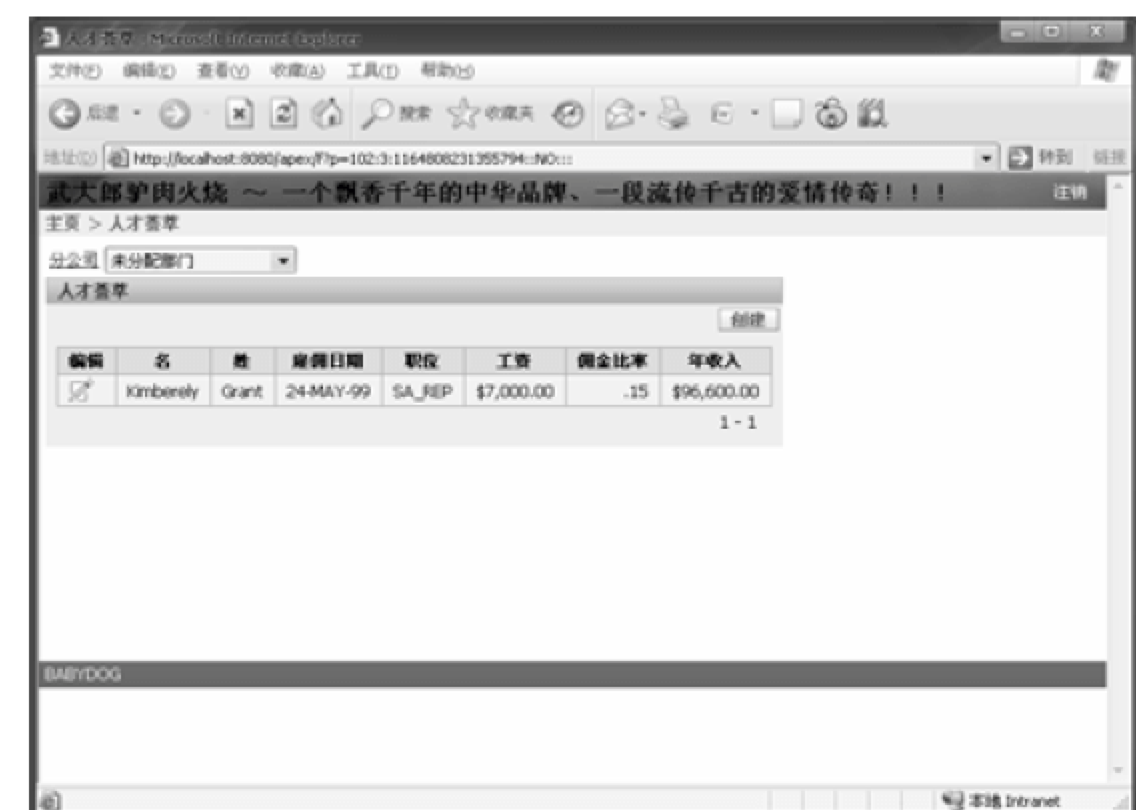


图　23.132

（7）单击第 3 行最左面的编辑图标编辑该行数据记录，如图 23.133 所示。

（8）此时，用户可以修改这行数据，也可以删除这行数据等。操作后单击“后退”按钮就可以退回到“人才荟萃”页面，如图 23.134 所示。



图　23.133

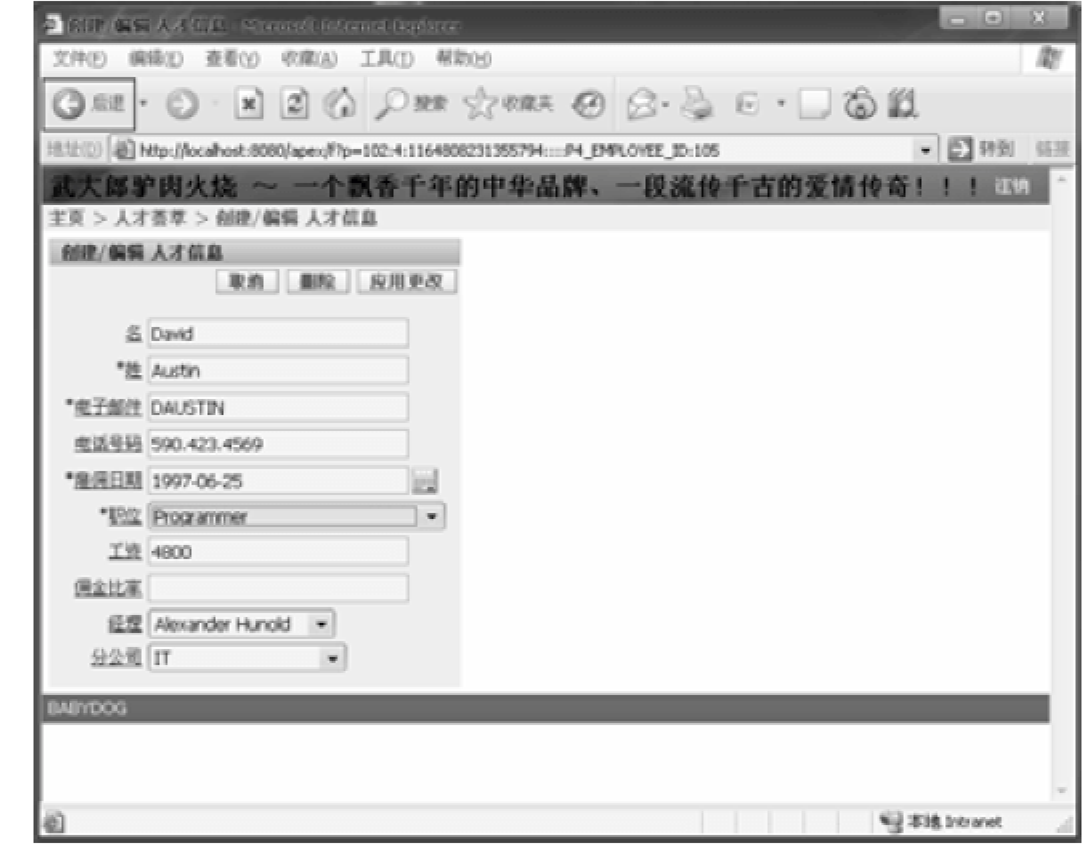


图　23.134

（9）单击“创建”按钮，插入一行新的数据，如图 23.135 所示。

（10）此时，用户就可以输入相关的人才信息以插入新的数据行，如图 23.136 所示。最后完成了所有操作之后，您就可以退出该应用程序。



图　23.135

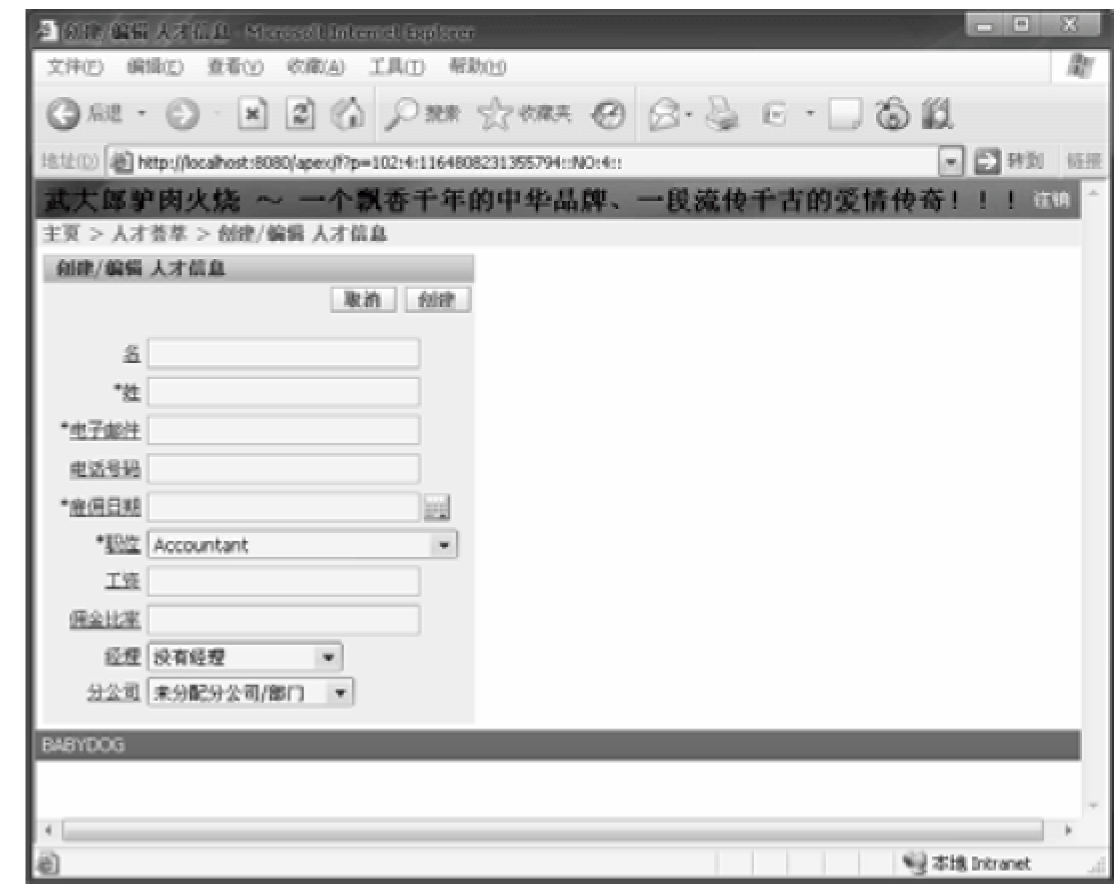


图　23.136

通过以上的操作，您终于可以确信所开发的应用程序完全满足了公司的要求。用了这么短的时间就开发出大型软件公司声称需要几十个高级程序员开发一年才能完成的应用系统，是不是自己也觉得自己确实是一个 IT 方面的奇才呢？

老板对您在这么短的时间里为公司创造了几百万个驴肉火烧的价值这一事实是看在眼里，喜在心头。为了表彰您的卓越贡献，老板还专门为您的提职问题召开了一次董事会。

目前武大郎烧饼总店的技术职位分为见习烙饼师、正式烙饼师、铜牌烙饼师、银牌烙饼师、金牌烙饼师和白金烙饼师。

老板觉得即使提升您为最高的技术职位白金烙饼师也无法体现您的价值。为此公司决定再增设一个更高的职位钻石烙饼师，并立即将您破格晋升为钻石烙饼师，并要求人事部门马上将这一新职称加到您的名片上，虽然您根本不会烙饼。这也应了那句老话，什么事也不会干的，一定是干大事的，您也成了一位干大事的人物。

参 考 文 献

1．Amoroso E. G. Fundamentals of Computer Security Technology. New Jersey, USA: P T R Prentice Hall, Inc, 1994

2．Austin D., & Dyke R. van el. (2001). Oracle9i New Features for Administrators: Student Guide Volume 1 & 2. USA: Oracle Corporation

3．Austin D., & Ernst B. el. (1999). Oracle8i New Features for Administrators: Instructor Guide Volume 1-3. USA: Oracle Corporation

4．Barker R.(1994). CASE*METHOD: Entity Relationship Modelling. USA: R.R. Donnelley & Sons Company

5．Chaitanya K. (2007). Oracle Database 11g: SQL Fundamentals II. USA: Oracle Corporation

6．Chon S. & Green R. (1999). Data Warehousing Fundamentals: Student Guide Volume 1 & 2. USA: Oracle Corporation

7．Couchman J. S. (2002). OCP Oracle9i SQL 考试指南（英文版）．北京：机械工业出版社

8．Drew A. (2008). Oracle Database SQL Language Reference, 11g Release 1 (11.1). USA: Oracle Corporation

9．Drue B. (2009). Oracle Application Express SQL Workshop and Utilities Guide, Release 3.2. USA: Oracle Corporation

10．Ernest B. & Rasmussen H. R. (1999). Enterprise DBA Part 1A: Architecture and Administration: Student Guide Volume 1 & 2. USA: Oracle Corporation

11．Gossett S. & Jang S. (1999). Oracle8 Advanced Replication: Instructor Guide Volume 1&2. USA: Oracle Corporation

12．Greenberg N. & Nathan P. (2001). Introduction to Oracle9i: SQL: Student Guide Volume 1 & 2. USA: Oracle Corporation

13．Joel K. & Sharon K. (2009). Oracle Database 2 Day + Application Express Developer's Guide, Release 3.2. USA: Oracle Corporation

14．Kochhar N. & Gravina E. (1998). Introduction to Oracle: SQL and PL/SQL: Student Guide Volume 1 & 2. USA: Oracle Corporation

15．Kochhar N. & Kramer D. (1996). Introduction to Oracle: SQL and PL/SQL Using Procedure Builder: Participant Guide Volume 1 - 4. USA: Oracle Corporation

16．Lorentz D. (2000). SQL Reference Release 3 (8.1.7). USA: Oracle Corporation

17．Lorentz D. (2001). Oracle9i SQL Reference Release 1 (9.0.1). USA: Oracle Corporation

18．Marco A. (2009). Oracle Application Express Application Builder User's Guide, Release 3.2. USA: Oracle Corporation

19．Marco A. (2009). Oracle Application Express Installation Guide, Release 3.2. USA: Oracle Corporation

20．Nancy G. (2004). Oracle Database 10g: SQL Fundamentals I. USA: Oracle Corporation

21．Priya V. (2004). Oracle Database 10g: SQL Fundamentals II. USA: Oracle Corporation

22．Priya V. (2004). Oracle Database 10g: SQL Tuning Workshop. USA: Oracle Corporation

23．Puja S. (2007). Oracle Database 11g: SQL Fundamentals I. USA: Oracle Corporation

24．Rob P. & Coronel C. (1993). Database Systems Design, Implementation, and Management. Belmont, California: Wadsworth Publishing Company

25．Schwinn U. & Venkatachalam V. (1998). Oracle8 Database Administration: Student Guide Volume 1 - 3. USA: Oracle Corporation

26．Shashaanka A. (2005). Oracle Database PL/SQL User's Guide and Reference 10g Release 2 (10.2). USA: Oracle Corporation

27．Sundeep A. (2005). Oracle Database SQL Reference, 10g Release 2 (10.2). USA: Oracle Corporation

28．Terri J. (2009). Oracle Application Express Administration Guide, Release 3.2. USA: Oracle Corporation

29．Tone T. (2005). Oracle Database 10g: Develop Applications Using HTML DB Ed2. USA: Oracle Corporation

30．Tulika S. (2006). Oracle Database 10g: Develop PL/SQL Program Units. USA: Oracle Corporation

31．Tulika S. (2007). Oracle Database 11g: PL/SQL Fundamentals. USA: Oracle Corporation

32．Watt S. (2001). iSQL*Plus User's Guide and Reference Release 9.0.1. USA: Oracle Corporation

33．Watt S. (2001). SQL*Plus Getting Started Release 9.0.1 for Windows. USA: Oracle Corporation

34．Watt S. (2001). SQL*Plus User's Guide and Reference Release 9.0.1. USA: Oracle Corporation

35．何明．Oracle DBA 基础培训教程——从实践中学习 Oracle DBA．北京：清华大学出版社，2006

36．何明．Oracle DBA 培训教程——从实践中学习 Oracle 数据库管理与维护．北京：清华大学出版社，2009

结　束　语

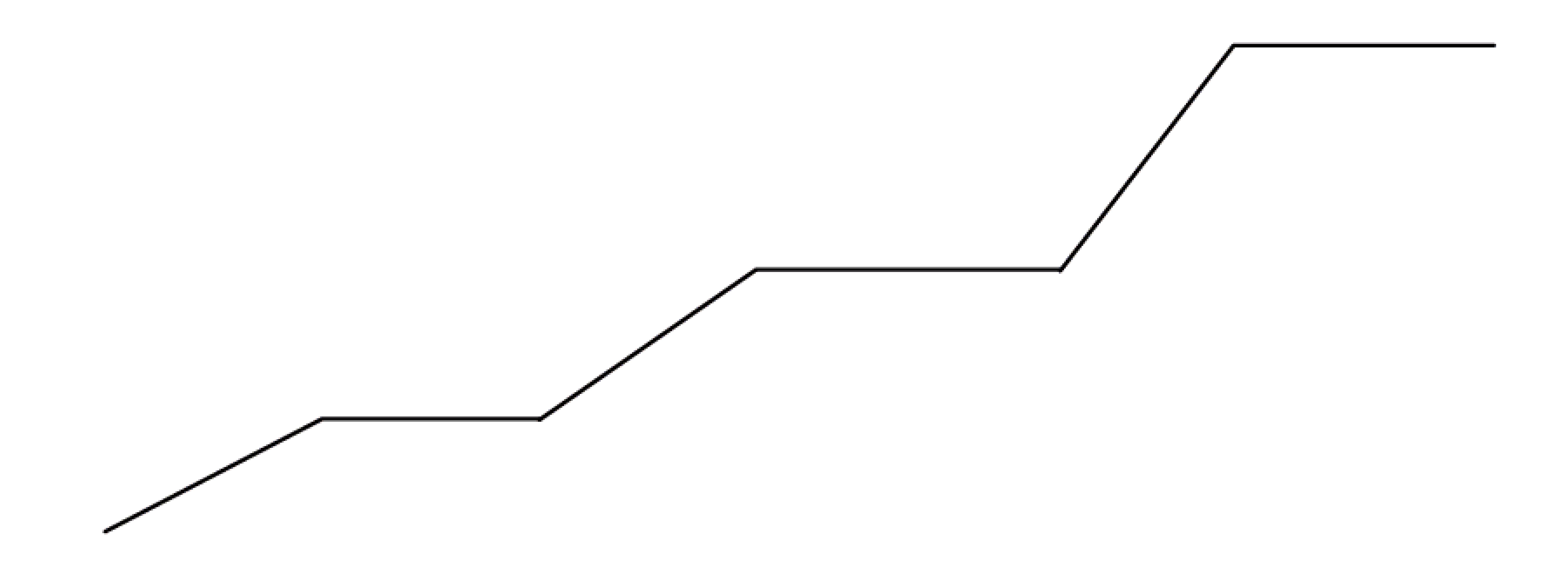
当您学完了这本书时有什么感想？相信您已成为了 Oracle SQL 的行家了吧！

您在读本书时可能遇到过这样的事情，做本书的例题时，可能使用不同的 SQL 语句来获得完全相同的结果。如果是这样的话，您不用担心，只要结果是正确的就行。还是那句老话“不管它白猫还是黑猫，抓住老鼠就是好猫”。还记得是谁说的吗？

通常要熟练地掌握一门能保住饭碗的手艺（技能），需要较长的时间反复地练习才行。您根本不可能通过参加几天 Oracle SQL 的培训课程就成为这方面的行家。这种课程只能达到把您引入 Oracle 这个行当中的目的（希望如此）。之后，您还需要自己做大量的练习才能熟练地掌握 Oracle SQL 的使用。所谓“师傅领进门，修行靠个人”，帮助读者“修行”Oracle SQL 在实际工作中的应用正是这本书的主要目的。如果读者没有在工作中使用过 SQL 或使用得不多，希望您能多花点时间把本书上的大部分例题自己在计算机上再做上 1~2 遍。如果您没有计算机，您就要更仔细地阅读例题和例题结果，并把这些结果想象成计算机终端的输出（实际上本书上的例题和例题结果都是从计算机屏幕上复制的）。

我个人觉得学习 Oracle 的使用就像学习任何其他手艺（技能）一样，如剑法，您无论是练避邪剑法还是练独孤九剑，您都一定要练。您要通过大量和反复的练习才能悟出其中的奥秘，才能上到一个新的层次。只有这样，在遇到了实际问题时，您才能随心所欲地使用 SQL 语句。我不相信学习 Oracle 可以速成，速成最多教会您会说，绝不可能使您达到会做的程度。

如果您在阅读此书时有个别的内容无论怎样冥思苦想也不能理解，这也是很正常的。但您一定要坚持把这本书读完。下面的图可能会帮助您理解学习的规律。



当您在没有学完某一门课时，就相当于在斜坡上。此时如果停下来，您就会很快地掉到开始学习的起点。如果学完了这门课，您就相当于在平台上。这时您就不容易退步了。所以一旦开始了本书的学习，最好是坚持下去把它学完。当站在了高高的平台上回过头往

下看时，许多当初困扰您的问题就变得很简单了。

培训与高等教育是有所不同的。可能许多读者看过武侠电视或小说。培训可能像教您如何练剑，高等教育可能像教您如何练气（功）。如果您学会了上乘剑法，即使内功不怎么样，也照样可以在险恶的（职业）江湖中闯荡。同样，如果您真能熟练地使用 Oracle 并能解决实际问题，即使您说不出其中的原因，公司照样把您当个宝。绝大多数经理或老板们对造成数据库故障的原因根本不感兴趣（可能根本就不懂），他们感兴趣的是在数据库出现故障时谁能以最快的速度修好它而且最好不丢失数据。

有人认为 Oracle 这个职业像一叶方舟，也有人认为它是条贼船。

认为它是方舟的原因可能是在目前经济全球化、就业市场的竞争异常激烈的严酷现实面前，Oracle 专业人员的就业机会要远远高于绝大多数其他的行业，而且它的工资之高也是绝大多数其他行业的从业者们所垂涎的。

认为它是条贼船的原因可能是它的门坎较高，培训费用很高，学习的时间也较长。仅以 Oracle SQL 为例，如果您是一个真正的初学者，不花上 1~2 个月，也许更长的时间练习，是很难在实际工作中熟练地使用 SQL 的。也许您在网上看到有人用了 1~2 周就考过了 Oracle SQL 这部分的 OCP 考试，如果他/她真的是初学者，而且不是通过背真题考过的话，那么他/她可能不是人，可能是哪个“天神转世”，如孙悟空或猪八戒，而且我相信他/她很可能根本不会使用 SQL，即只会说不会干。

另一个认为它是条贼船的原因可能是，要想呆在这条贼船上并非易事。因为 Oracle 平均每两年左右就要升级一次，也就是持有 OCP 证书的人可能需要参加升级考试。所谓“上贼船难，要想呆在贼船上就更难”。

我认为这里多多少少有点误会。根据我个人的经验和从同行那里得来的信息，发现了如下的规律：一般人在加入 Oracle 这个行当时，确实要下一番工夫。第 1 个 OCP 证书拿起来的确不轻松。一旦您入了行，并在这一行中工作了一段时间，升级对您已不是个问题了。其实，从 Oracle 7.x 版本到现在的 Oracle 9i，其体系结构变化很小，而且 Oracle SQL 的基本语句也几乎没什么变化。据我所知，许多企业在招聘 Oracle 专业人员时只关心应征者有没有证书，至于是哪个版本的他们并不关心。可能就像某个职位要求应征者必须大学毕业，至于哪年毕业的就不重要了。许多企业真正关心的是您是否真的具有 Oracle 方面的实际工作经验。还是那句老话，老板需要的是能为他干活的人。而要能干活，您就一定要做大量的上机练习。

从实用的角度来看，方舟和贼船实际上都差不多。即不管它是方舟还是贼船，先上了船再说，因为这样做，总比在波涛汹涌的职海当中沉下去给龙王爷当女婿强（长期失业）。

再强调一下，真正检验读者能力的是市场，而不是考场，企业需要的是会干活的员工。

如果读者对本书有任何意见或要求，欢迎来信提出。我的电子邮箱为：sql_minghe@yahoo. com.cn。

最后，恭祝读者胜利地完成 Oracle SQL 的学习之旅。

读者意见反馈卡

您购买的书名：________________　您的姓名：____________________　性　别：☐男　☐女
年龄：________________________　文化程度：____________________　职　业：______________
邮编：________________________　通信地址：____________________　E-mail：______________
您常用的软件：1 ______________ 2 ______________ 3 ______________ 4 ______________

您购买本书的原因（可多选）：

☐封面与装帧　☐引言目录　☐正文内容　☐丛书风格　☐价格　☐光盘　☐专业性强　☐别人介绍
☐出版社或作者名声　☐售后服务

本书最令您满意的是（可多选）：

☐专业性强、覆盖面广　☐内容翔实、定位准确　☐精益求精、售后服务

您可以承受的图书价格：

☐20 元以下　☐30 元以下　☐40 元以下　☐50 元以下　☐只要内容好，不论价格

您对本书的评价：

封面装帧：☐很好　☐较好　☐一般　☐不满意　建议______________________________
印刷质量：☐很好　☐较好　☐一般　☐不满意　建议______________________________
正文质量：☐很好　☐较好　☐一般　☐不满意　建议______________________________
写作风格：☐很好　☐较好　☐一般　☐不满意　建议______________________________
专业水平：☐很好　☐较好　☐一般　☐不满意　建议______________________________

您希望增加哪些图书选题： 1________________ 2 ________________ 3____________

您认为本书有哪些错误：

章______节______页码______行______列_______　图号______　错误__________应改为 __________
章______节______页码______行______列_______　图号______　错误__________应改为 __________
章______节______页码______行______列_______　图号______　错误__________应改为 __________
章______节______页码______行______列_______　图号______　错误__________应改为 __________

您的其他建议：

1 ____
2 ____
3 ____

请填好本卡后寄给：
清华大学校内金地公司　邮编：100084　电话：（010）62788951-221
《Oracle SQL 培训教程——从实践中学习 Oracle SQL 及 Web 快速应用开发》编辑部收
传真：（010）62788903　公司网址：www.thjd.com.cn　E-mail：liulm@vip.163.com
如需本书可与本编辑部联系邮购，汇款请按以上地址填写，另加邮费 15%（挂号）